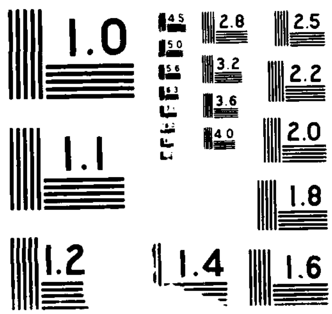
AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION 1/6
SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA
B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317
UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

IDA MEMORANDUM REPORT M-460

AN Ada/SQL APPLICATION SCANNER

Bill R. Brykczynski
Fred Friedman
Kevin Heatwole
Kerry Hilliard

March 1988

Prepared for
Office of the Under Secretary of Defense for Research and Engineering

IDA
INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE AD-A194517

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Public release/unlimited distribution.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Memorandum Report M-460	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION OUSDA, DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION WIS JPMO/XPT 8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) Washington, D.C. 20330-6600	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-W5-206 / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
An Ada/SQL Application Scanner (U)

12 PERSONAL AUTHOR(S)
Bill R. Brykczynski, Fred Friedman, Kevin Heatwole, Kerry Hilliard

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM _____ TO _____	1988 March	570

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Ada Programming Language; Structured Query Language (SQL); Software Tools; Ada/SQL; Data Definition Language (DDL); Data Manipulation Language (DML); Oracle Database Management Systems (DBMS); Interface Software; WIS; VAX/VMS; DEC Ada Compiler.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)
This IDA Memorandum Report identifies and describes a version of software delivery, the Ada/SQL Application Scanner. The purpose of this software system is to provide a tool which will aid in the generation of subprograms necessary for a Level 1 Ada/SQL implementation. Ada/SQL is an interface between the Ada programming language and the database programming language SQL. Ada/SQL, like SQL, comprises two main components: a Data Definition Language (DDL) and a Data Manipulation Language (DML). Both the DDL and the DML are coded using pure Ada syntax and semantics. The DDL resides in a package specification and is used to define the data types, variable definitions, and table and column definitions. The DML is expressed as syntax very similar to the syntax of SQL DML. This expression is allowed due to a set of underlying operators and subprograms which must be 'with'ed by the application. However, many of these subprograms are application dependent and are tedious to code. To alleviate the coding of these subprograms, a tool, the Application Scanner, has been developed. The Application Scanner reads the Ada/SQL data definition package, the Ada/SQL data manipulation package, and various other packages to determine exactly the necessary functions and procedures required for compilation. If errors are found in any of

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ■ UNCLASSIFIED/UNLIMITED □ SAME AS RPT. □ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Bill R. Brykczynski	22b TELEPHONE (Include Area Code) (703) 824-5515 22c OFFICE SYMBOL IDA/CSED

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. (continued)

these files, a listing will be generated showing the text of the package, and pointer to the appropriate line number and column position where the error occurred.

If no errors were detected by the Application Scanner, a package will be generated containing subprogram definitions which must be then compiled. This specific instance of the Application Scanner generates subprograms which access the Oracle database management system.

IDA MEMORANDUM REPORT M-460

AN Ada/SQL APPLICATION SCANNER

Bill R. Brykczynski
Fred Friedman
Kevin Heatwole
Kerry Hilliard

March 1988

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-W5-206

CONTENTS

1. SCOPE

1.1 Identification

The purpose of this IDA Memorandum Report is to identify and describe a version of a software delivery, "An Ada/SQL Application Scanner," to the WIS Joint Program Management Office. The term *version* is applied to the initial release as well as to all interim changes. This report was written to describe the software developed to satisfy deliverable 5.c of task order T-W5-206, entitled WIS Application Software Study.

1.2 System Overview

The purpose of this software system is to provide a tool which will aid in the generation of subprograms necessary for a Level 1 Ada/SQL implementation. Ada/SQL is an interface between the programming language Ada [ADA 83] and the database programming language SQL [SQL 86]. Ada/SQL, like SQL, is comprised of two main components: a Data Definition Language (DDL) and a Data Manipulation Language (DML). Both the DDL and the DML are coded using pure Ada syntax and semantics. The DDL resides in a package specification, and is used to define the data types, variable definitions, and table and column definitions. The DML is expressed as syntax very similar to the syntax of SQL DML. This expression is allowed due to a set of underlying operators and subprograms which must be 'with'ed by the application. However, many of these subprograms are application dependent and are tedious to code. To alleviate the coding of these subprograms, a tool, named the application scanner, has been developed.

The application scanner reads the Ada/SQL data definition package, the Ada/SQL data manipulation package, and various other packages to determine exactly the necessary functions and procedures required for compilation. If errors are found in any of these files, a listing will be generated showing the text of the package, and pointer to the appropriate line number and column position where the error occurred.

If no errors were detected by the application scanner, a package will be generated containing subprogram definitions which must then be compiled. This specific instance of the application scanner generates subprograms which access the database management system Oracle®. Additional code necessary to access Oracle is found in [IDA 88]. The Level 1 Ada/SQL definition can be found in [IDA 87].

1.3 Documentation Overview

The file [BBRYKCZYN.EXAMPLE]READ.ME is included with the magnetic tape containing the interface software. This file contains guidelines which show the user how to create an Ada/SQL application, use the application scanner, and in what order to compile the output from the scanner.

ORACLE is a registered trademark of Oracle Corporation

The directory located in [BBRYKCZYN.EXAMPLE] provides a comprehensive example of using the Ada/SQL system, connected to Oracle, and using the application scanner.

2. REFERENCED DOCUMENTS

[ADA 83] U.S. Department of Defense. 1983. *ANSI/MIL-STD-1815A, Military standard: Ada programming language.* Washington, D.C.: U.S DoD.

[ANSI 86] American National Standards Institute. 1986. *ANSI X3.135.1986, Database language SQL.* New York: ANSI.

[IDA 87] Brykczynski, Bill, Fred Friedman, and Kerry Hilliard. 1988. *Level 1 Ada/SQL database language interface user's guide.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-361.

[IDA 88] Brykczynski, Bill, Fred Friedman, and Kerry Hilliard. 1988. *An Oracle-Ada/SQL implementation.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-459.

3. VERSION DESCRIPTION

3.1 Inventory of Materials Released

This prototype Ada/SQL system was developed on a VAX™ 8600, using VAX/VMS version 4.6, and the DEC Ada compiler, version 1.4-33. The magnetic tape upon which the source is located is in VAX/VMS backup format, and the save set name is ADASQL. This tape requires 8192 blocks of memory. To load the tape, allocate the tape drive desired, request a tape mount, and issue the following command: "BACKUP MUA0: [appropriate directory...]*.*.*", where MUA0 is the logical tape drive name, and appropriate directory is the directory in which you will be placing the contents of the tape.

3.2 Inventory of Software Prototype Contents

The following are the files which make up the prototype Ada/SQL system. They are listed in compilation order. There are two naming conventions used. First, a major portion of the application scanner uses a tool called the ddl reader. Package specifications are suffixed by "spec.ada", and package bodies are suffixed by ".ada" for ddl reader code. For the remaining portions of code, package specifications are suffixed by "s.ada", and package bodies are suffixed by "b.ada". All of the files use an abbreviated name for the physical file name derived from the name of the package.

funcdefs.ada
ddl_database.ada
ddl_keyword_spec.ada
ddl_keyword.ada
txtprts.ada
txtprtb.ada
lexs.ada
lexb.ada
ddl_io_defs_spec.ada
ddl_io_defs.ada
ddl_defs.ada
ddl_new_des_spec.ada
ddl_extra_defs.ada
enums.ada
enumb.ada
dummys.ada
ddl_variables.ada
columns.ada
columnb.ada
withs.ada
withb.ada

VAX is a trademark of Digital Equipment Corporation

results.ada
resultb.ada
indexs.ada
indexb.ada
dbtypes.ada
dbtypeb.ada
comptos.ada
comptob.ada
chartos.ada
chartob.ada
tables.ada
tableb.ada
pdtypes.ada
pdtypeb.ada
ddl_add_des_spec.ada
ddl_add_des.ada
unquals.ada
unqualb.ada
quals.ada
qualb.ada
corrs.ada
corrb.ada
convs.ada
convb.ada
intos.ada
intob.ada
pgmconvs.ada
pgmconvb.ada
predefs.ada
predefb.ada
froms.ada
fromb.ada
clauses.ada
clauseb.ada
indics.ada
indicb.ada
genfuncs.ada
genfuncb.ada
selecs.ada
selecb.ada
names.ada
nameb.ada
semans.ada
semanb.ada
posts.ada
postb.ada
syntacs.ada
syntacb.ada
tents.ada

tentb.ada
exprs.ada
exprb.ada
ddl_schema_io_errors_spec.ada
scans.ada
searchs.ada
statements.ada
tblexprs.ada
selects.ada
selectb.ada
statementb.ada
searchb.ada
tblexprb.ada
ddl_schema_io_internal_spec.ada
ddl_schema_io_spec.ada
ddl_new_des.ada
ddl_schema_io.ada
ddl_subroutines_1_spec.ada
ddl_subroutines_1.ada
ddl_show_spec.ada
ddl_show.ada
ddl_schema_io_internal.ada
ddl_schema_io_errors.ada
ddl_end_spec.ada
ddl_end.ada
ddl_search_des_spec.ada
ddl_search_des.ada
ddl_error_spec.ada
ddl_error.ada
ddl_use_spec.ada
ddl_use.ada
ddl_subroutines_2_spec.ada
ddl_subroutines_4_spec.ada
ddl_subroutines_4.ada
ddl_subroutines_3_spec.ada
ddl_subroutines_3.ada
ddl_names_spec.ada
ddl_names.ada
ddl_with_spec.ada
ddl_with.ada
ddl_auth_spec.ada
ddl_auth.ada
ddl_function_spec.ada
ddl_function.ada
ddl_subroutines_2.ada
ddl_package_spec.ada
ddl_package.ada
ddl_list_spec.ada
ddl_list.ada

ddl_integer_spec.ada
ddl_integer.ada
ddl_float_spec.ada
ddl_float.ada
ddl_enumeration_spec.ada
ddl_enumeration.ada
ddl_derived_spec.ada
ddl_derived.ada
ddl_variable_spec.ada
ddl_variable.ada
ddl_subtype_spec.ada
ddl_subtype.ada
ddl_record_spec.ada
ddl_record.ada
ddl_array_spec.ada
ddl_array.ada
ddl_type_spec.ada
ddl_type.ada
ddl_driver_spec.ada
ddl_driver.ada
ddl_call_to_ddl_spec.ada
ddl_call_to_ddl.ada
scanb.ada
main.ada

3.3 Changes

Not applicable.

3.4 Adaptation Data

Not applicable.

3.5 Interface Compatibility

Not applicable.

3.6 Bibliography of Reference Documents

Brykczynski, Bill, and Fred Friedman, *Preliminary version: Ada/SQL: A standard, portable Ada-DBMS interface.* Alexandria, VA: Institute for Defense Analyses, 1988. IDA Paper P-1944.

Brykczynski, Bill, and Fred Friedman, *Ada/SQL binding specifications,* Alexandria, VA: Institute for Defense Analyses, 1988. IDA Memorandum Report M-362.

Date, C.J., *A guide to the SQL standard.* New York: Addison-Wesley, 1987.

3.7 Summary of Changes

Not applicable.

3.8 Installation Instructions

To load the contents of the tape onto disk, allocate and mount a tape drive. Next, issue the following command: "BACKUP MUA0: [appropriate directory...]*.*.*" Where MUA0: is the name of the tape drive, and appropriate directory is the name of the directory the contents are to be loaded.

3.9 User Guidelines

The following is a set of guidelines for using the VAX/VMS Level 1 Ada/SQL connected to the Oracle database management system. These guidelines assume that a directory exists which contains the files loaded from tape. The files on the tape were loaded from a directory named [BBRYKCZYN.ORACLE]. Of course, when a tape is loaded on another system, this path name will be different.

1) Create the ADASQL$ENV logical

There are several files which are read by the application scanner to establish a predefined environment for processing application programs. These files are DATABASE.ADA, CURSOR_DEFINITION.ADA, and STANDARD.ADA. These files are not source files that are linked with the Ada/SQL application programs. They must, however, be stored in a directory that is accessible via the VAX/VMS logical name ADASQL$ENV whenever the application scanner is run. These files should not be compiled or otherwise used for any purpose other than that described here. To assign the logical, type in the following:

```
ASSIGN [BBRYKCZYN.ORACLE.STANDARDS] ADASQL$ENV
```

2) Copy over the AUTH_PACK.ADA file

In SQL, a module must contain an authorization identifier which identifies the user. In Ada/SQL, the authorization identifier must reside in a file called AUTH_PACK.ADA. At this time, it is necessary only to copy an AUTH_PACK.ADA from another directory and compile it into the library. A sample AUTH_PACK.ADA is located in directory [BBRYKCZYN.ORACLE.EXAMPLE]

3) Create the Ada/SQL application specific files

There are four files one must create in order to use Ada/SQL. These are the _TYPES.ADA, _VARIABLES.ADA, _DDL.ADA packages, and the main program. The files must be named exactly as the package name, with the addition of a '.ADA' suffix. Examples of these files are included in the [BBRYKCZYN.ORACLE.EXAMPLE] directory.

4) Create the Oracle DDL

It is necessary for Oracle to have the data definition exist prior to the running of an Ada/SQL program. If you are building an Ada/SQL program to access a pre-existing database definition, this step can be deleted. If you are building a new application, it will be necessary to invoke Oracle, and create the appropriate table and column definitions.

5) Run the scanner

To run the application scanner, type in the following command: "RUN [BBRYKCZYN.-ORACLE.APSCAN_SOURCE]APSCAN.EXE". The application scanner will prompt you with several questions:

a) **Enter DML filename:**

Here you enter the name of your Ada compilation unit which contains DML statements which you want processed by the application scanner. An output file will be generated where errors in the DDL will be reported. This file will have the name of your compilation unit's library name, suffixed with .DDLOUT. For example enter BILL.ADA here (the subprogram name is BILL) and any DDL errors will be listed in a file called BILL.DDLOUT.

b) **Enter listing filename:**

Here you enter the name of a file where the application scanner will report errors in the DML. For example if you had entered BILL.ADA for question one you could enter BILL.LST here. Only DML errors will be reported here, DML errors are in the *.DDLOUT file.

c) **Enter filename for generated functions:**

Here you enter the file name for the compilation unit which will contain the functions necessary to make your DML compilation unit compilable. This will be an Ada compilation unit which will become a part of your program. For example if you had entered BILL.ADA for question one you could enter BILL_ADA_SQL.ADA here. The library unit name for this compilation unit will be the library unit name of your compilation unit with an extension of _ADA_SQL. The subprogram name in BILL.ADA is BILL), and the library unit name of the compilation unit BILL_ADA_SQL.ADA will be BILL_ADA_SQL. Your compilation unit must

The application scanner will then notify you:

Invoking application scanner with:
DML filename => file name you entered in #1 above
Listing filename => file name you entered in #2 above
Generated package => file name you entered in #3 above

When the application scanner is complete it will issue the message:

%ADASQL-I-SCAN, Scan completed with errors
or the message:

%ADASQL-I-SCAN, Scan completed with no errors

In the case of 'with errors' check the *.DDLOUT file to make sure the DDL scanned correctly, then check the listing file you specified in #2 above to see if there was an error in the DML. Correct the errors and run the application scanner again. In the case of 'with no errors' you must still check the *.DDLOUT file. If errors are reported in this file but not in the listing file the message at the end of the application scanner will indicate no errors.

Repeat these steps until you have generated a function package through the application scanner for all your compilation units which contain DML. The package generated by the application scanner must be withed in your compilation unit.

6) Compile the output of the scanner

When a correct version of Ada/SQL DML is processed by the scanner, a generated package will be produced which must be compiled. This package contains the various subprograms which allow the Ada/SQL DML to interact with the database.

7) Compile and link the DML package

After compiling the generated *_ADA_SQL.ADA package from the previous step, the Ada/SQL DML package may now be compiled. Continuing with the example, this file is named BILL.ADA. After compiling, the file must be linked, which, in this example, results in an executable named BILL.

8) Execute the Application

3.10 Possible Problems and Known Errors

The following items are incorrectly processed by the application scanner, but are caught by the Ada compiler:

a) Package names form their own name space, so with'ed package names are not hidden by table, variable, or enumeration literal declarations. Also, homographs are not recognized, so if both packages A and B are with'ed and use'd, and B declares a type A, then type A becomes visible.

b) Enumeration literals form their own name space, so can be named the same as type/subtype names.

c) Declaring an enumeration subtype also makes the enumeration literals visible, e.g., package A declares enumeration type T, package B with's A and declares enumeration subtype S, package C with's only B but can use T's enumeration literals to declare a range constraint for a subtype of S.

3.11 Source Listings

3.11.1 package funcdefs.ada

```
-- funcdefs.ada - definitions of SQL operations

-- This is extracted from package ADA_SQL_FUNCTIONS of the runtime version,
-- which should eventually be updated to use this same definition package

package ADA_SQL_FUNCTION_DEFINITIONS is

  type SQL_OPERATION is
  ( O_AVG              , O_MAX         , O_MIN        , O_SUM               ,
    O_UNARY_PLUS       , O_UNARY_MINUS , O_PLUS       , O_MINUS             ,
    O_TIMES            , O_DIVIDE      , O_EQ         , O_NE                ,
    O_LT               , O_GT          , O_LE         , O_GE                ,
    O_BETWEEN          , O_AND         , O_IS_IN      , O_OR                ,
    O_NOT              , O_LIKE        , O_AMPERSAND  , O_SELEC             ,
    O_SELECT_DISTINCT  , O_ASC         , O_DESC       , O_TABLE_COLUMN_LIST ,
    O_COUNT_STAR       , O_NULL_OP     , O_STAR       , O_NOT_IN            ,
    O_VALUES           , O_DECLAR );

end ADA_SQL_FUNCTION_DEFINITIONS;
```

3.11.2 package ddl_database.ada

```
package DATABASE is
  type INT          is new STANDARD.INTEGER;
  type DOUBLE_PRECISION is new STANDARD.FLOAT;
  type CHAR         is new STANDARD.STRING;
  type CHAR_LINK is access CHAR;
  type USER_AUTHORIZATION_IDENTIFIER is new STANDARD.STRING;
  type USER_AUTHORIZATION_IDENTIFIER_LINK is access
       USER_AUTHORIZATION_IDENTIFIER;
  type COLUMN_NUMBER is new STANDARD.INTEGER;
end DATABASE;
```

3.11.3 package ddl_keyword_spec.ada

```
package KEYWORD_ROUTINES is

  function SQL_KEY_WORD
          (IN_STRING : in STRING)
           return BOOLEAN;

  function ADA_KEY_WORD
          (IN_STRING : in STRING)
           return BOOLEAN;

end KEYWORD_ROUTINES;
```

3.11.4 package ddl_keyword.ada

```
package body KEYWORD_ROUTINES is

------------------------------------------------------------------------
--
-- table of the SQL key words which cannot be used as identifiers

  type KEYWORD_POINTER is access STRING;
  type KEYWORD_ARRAY is array (INTEGER range <>) of KEYWORD_POINTER;
  SQL_KEYWORDS : constant KEYWORD_ARRAY := (
                 new STRING' ("ALL"),
                 new STRING' ("AND"),
                 new STRING' ("ANY"),
                 new STRING' ("AS"),
                 new STRING' ("ASC"),
                 new STRING' ("AUTHORIZATION"),
                 new STRING' ("AVG"),
                 new STRING' ("BEGIN"),
                 new STRING' ("BETWEEN"),
                 new STRING' ("BY"),
                 new STRING' ("CHAR"),
                 new STRING' ("CHARACTER"),
                 new STRING' ("CHECK"),
                 new STRING' ("CLOSE"),
                 new STRING' ("COBOL"),
                 new STRING' ("COMMIT"),
                 new STRING' ("CONTINUE"),
                 new STRING' ("COUNT"),
                 new STRING' ("CREATE"),
                 new STRING' ("CURRENT"),
--                 new STRING' ("CURSOR"),
                 new STRING' ("DEC"),
                 new STRING' ("DECIMAL"),
                 new STRING' ("DECLARE"),
                 new STRING' ("DELETE"),
                 new STRING' ("DESC"),
                 new STRING' ("DISTINCT"),
                 new STRING' ("DOUBLE"),
                 new STRING' ("END"),
                 new STRING' ("ESCAPE"),
                 new STRING' ("EXEC"),
                 new STRING' ("EXISTS"),
                 new STRING' ("FETCH"),
                 new STRING' ("FLOAT"),
                 new STRING' ("FOR"),
                 new STRING' ("FORTRAN"),
                 new STRING' ("FOUND"),
                 new STRING' ("FROM"),
```

```
new STRING' ("GO"),
new STRING' ("GOTO"),
new STRING' ("GRANT"),
new STRING' ("GROUP"),
new STRING' ("HAVING"),
new STRING' ("IN"),
new STRING' ("INDICATOR"),
new STRING' ("INSERT"),
new STRING' ("INT"),
new STRING' ("INTEGER"),
new STRING' ("INTO"),
new STRING' ("IS"),
new STRING' ("LANGUAGE"),
new STRING' ("LIKE"),
new STRING' ("MAX"),
new STRING' ("MIN"),
new STRING' ("MODULE"),
new STRING' ("NOT"),
new STRING' ("NULL"),
new STRING' ("NUMERIC"),
new STRING' ("OF"),
new STRING' ("ON"),
new STRING' ("OPEN"),
new STRING' ("OPTION"),
new STRING' ("OR"),
new STRING' ("ORDER"),
new STRING' ("PASCAL"),
new STRING' ("PLI"),
new STRING' ("PRECISION"),
new STRING' ("PRIVILEGES"),
new STRING' ("PROCEDURE"),
new STRING' ("PUBLIC"),
new STRING' ("REAL"),
new STRING' ("ROLLBACK"),
new STRING' ("SCHEMA"),
new STRING' ("SECTION"),
new STRING' ("SELECT"),
new STRING' ("SET"),
new STRING' ("SMALLINT"),
new STRING' ("SOME"),
new STRING' ("SQL"),
new STRING' ("SQLCODE"),
new STRING' ("SQLERROR"),
new STRING' ("SUM"),
new STRING' ("TABLE"),
new STRING' ("TO"),
new STRING' ("UNION"),
new STRING' ("UNIQUE"),
new STRING' ("UPDATE"),
```

```
                    new STRING' ("USER"),
                    new STRING' ("VALUES"),
                    new STRING' ("VIEW"),
                    new STRING' ("WHENEVER"),
                    new STRING' ("WHERE"),
                    new STRING' ("WITH"),
                    new STRING' ("WORK") );

-----------------------------------------------------------------------
--
-- table of the ADA key words which cannot be used as identifiers

  ADA_KEYWORDS : constant KEYWORD_ARRAY := (
                    new STRING' ("ABORT"),
                    new STRING' ("ABS"),
                    new STRING' ("ACCEPT"),
                    new STRING' ("ACCESS"),
                    new STRING' ("ALL"),
                    new STRING' ("AND"),
                    new STRING' ("ARRAY"),
                    new STRING' ("AT"),
                    new STRING' ("BEGIN"),
                    new STRING' ("BODY"),
                    new STRING' ("CASE"),
                    new STRING' ("CONSTANT"),
                    new STRING' ("DECLARE"),
                    new STRING' ("DELAY"),
                    new STRING' ("DELTA"),
                    new STRING' ("DIGITS"),
                    new STRING' ("DO"),
                    new STRING' ("ELSE"),
                    new STRING' ("ELSIF"),
                    new STRING' ("END"),
                    new STRING' ("ENTRY"),
                    new STRING' ("EXCEPTION"),
                    new STRING' ("EXIT"),
                    new STRING' ("FOR"),
                    new STRING' ("FUNCTION"),
                    new STRING' ("GENERIC"),
                    new STRING' ("GOTO"),
                    new STRING' ("IF"),
                    new STRING' ("IN"),
                    new STRING' ("IS"),
                    new STRING' ("LINITED"),
                    new STRING' ("LOOP"),
                    new STRING' ("MOD"),
                    new STRING' ("NEW"),
                    new STRING' ("NOT"),
                    new STRING' ("NULL"),
```

```
                new STRING' ("OF"),
                new STRING' ("OR"),
                new STRING' ("OTHERS"),
                new STRING' ("OUT"),
                new STRING' ("PACKAGE"),
                new STRING' ("PRAGMA"),
                new STRING' ("PRIVATE"),
                new STRING' ("PROCEDURE"),
                new STRING' ("RAISE"),
                new STRING' ("RANGE"),
                new STRING' ("RECORD"),
                new STRING' ("REM"),
                new STRING' ("RENAMES"),
                new STRING' ("RETURN"),
                new STRING' ("REVERSE"),
                new STRING' ("SELECT"),
                new STRING' ("SEPARATE"),
                new STRING' ("SUBTYPE"),
                new STRING' ("TASK"),
                new STRING' ("TERMINATE"),
                new STRING' ("THEN"),
                new STRING' ("TYPE"),
                new STRING' ("USE"),
                new STRING' ("WHEN"),
                new STRING' ("WHILE"),
                new STRING' ("WITH"),
                new STRING' ("XOR") );

-----------------------------------------------------------------------
--
-- SQL_KEY_WORD
--
-- return true if the string is a sql key word, false if not

  function SQL_KEY_WORD
       (IN_STRING : in STRING)
        return BOOLEAN is
  begin
    for I in SQL_KEYWORDS'RANGE loop
      if SQL_KEYWORDS (I).all = IN_STRING then
        return TRUE;
      end if;
    end loop;
    return FALSE;
  end SQL_KEY_WORD;

-----------------------------------------------------------------------
--
-- ADA_KEY_WORD
```

```
--
-- return true if the string is an ada key word, false if not

  function ADA_KEY_WORD
       (IN_STRING : in STRING)
        return BOOLEAN is
  begin
    for I in ADA_KEYWORDS'RANGE loop
      if ADA_KEYWORDS (I).all = IN_STRING then
        return TRUE;
      end if;
    end loop;
    return FALSE;
  end ADA_KEY_WORD;

end KEYWORD_ROUTINES;
```

3.11.5 package txtprts.ada

```
-- txtprts.ada - print utilities

with TEXT_IO;
  use TEXT_IO;
package TEXT_PRINT is

  type LINE_TYPE is limited private;

  type BREAK_TYPE is (BREAK, NO_BREAK);

  type PHANTOM_TYPE is private;

  procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE);

  procedure SET_LINE(LINE : in LINE_TYPE);

  function CURRENT_LINE return LINE_TYPE;

  procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL);
  procedure SET_INDENT(INDENT : in NATURAL);

  procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
                                    INDENT : in INTEGER);
  procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER);

  function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE;

  procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                         START_PHANTOM,
                         END_PHANTOM   : in PHANTOM_TYPE);
```

```
procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE);

procedure PRINT(FILE : in FILE_TYPE;
                LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure PRINT_LINE(FILE : in FILE_TYPE);
procedure PRINT_LINE(LINE : in LINE_TYPE);
procedure PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure BLANK_LINE(FILE : in FILE_TYPE);
procedure BLANK_LINE(LINE : in LINE_TYPE);
procedure BLANK_LINE;

generic
  type NUM is range <>;
package INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);

  procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

end INTEGER_PRINT;

generic
  type NUM is digits <>;
package FLOAT_PRINT is
```

```
    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);

    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

  end FLOAT_PRINT;

  NULL_PHANTOM : constant PHANTOM_TYPE;

  LAYOUT_ERROR : exception renames TEXT_IO.LAYOUT_ERROR;

private

  type PHANTOM_TYPE is access STRING;

  type LINE_REC(LENGTH : INTEGER) is
    record
      USED_YET              : BOOLEAN := FALSE;
      INDENT                : INTEGER := 0;
      CONTINUATION_INDENT   : INTEGER := 2;
      BREAK                 : INTEGER := 1;
      INDEX                 : INTEGER := 1;
      DATA                  : STRING(1..LENGTH);
      START_PHANTOM,
      END_PHANTOM           : PHANTOM_TYPE := NULL_PHANTOM;
    end record;

  type LINE_TYPE is access LINE_REC;

  NULL_PHANTOM : constant PHANTOM_TYPE := new STRING'("");

end TEXT_PRINT;
```

3.11.6 package txtprtb.ada

```
-- txtprtb.ada - print utilities

package body TEXT_PRINT is

  DEFAULT_LINE : LINE_TYPE;
```

```
procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE) is
begin
  LINE := new LINE_REC(LENGTH);
end CREATE_LINE;

procedure SET_LINE(LINE : in LINE_TYPE) is
begin
  DEFAULT_LINE := LINE;
end SET_LINE;

function CURRENT_LINE return LINE_TYPE is
begin
  return DEFAULT_LINE;
end CURRENT_LINE;

procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL) is
begin
  if INDENT >= LINE.LENGTH then
    raise LAYOUT_ERROR;
  end if;
  if LINE.INDEX = LINE.INDENT + 1 then
    for I in 1..INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
    LINE.INDEX := INDENT + 1;
  end if;
  LINE.INDENT := INDENT;
end SET_INDENT;

procedure SET_INDENT(INDENT : in NATURAL) is
begin
  SET_INDENT(DEFAULT_LINE,INDENT);
end SET_INDENT;

procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
                                  INDENT : in INTEGER) is
begin
  if LINE.INDENT + INDENT >= LINE.LENGTH or else LINE.INDENT + INDENT < 0
      then
    raise LAYOUT_ERROR;
  end if;
  LINE.CONTINUATION_INDENT := INDENT;
end SET_CONTINUATION_INDENT;

procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER) is
begin
  SET_CONTINUATION_INDENT(DEFAULT_LINE,INDENT);
end SET_CONTINUATION_INDENT;
```

```
function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE is
begin
  return new STRING'(S);
end MAKE_PHANTOM;

procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                       START_PHANTOM,
                       END_PHANTOM   : in PHANTOM_TYPE) is
begin
  LINE.START_PHANTOM := START_PHANTOM;
  LINE.END_PHANTOM := END_PHANTOM;
end SET_PHANTOMS;

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE) is
begin
  SET_PHANTOMS(DEFAULT_LINE,START_PHANTOM,END_PHANTOM);
end SET_PHANTOMS;

procedure PRINT(FILE : in FILE_TYPE;
                LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
  NEW_BREAK, NEW_INDEX : INTEGER;
begin
  if LINE.INDEX + ITEM'LENGTH + LINE.END_PHANTOM'LENGTH > LINE.LENGTH + 1
      then
    if LINE.INDENT + LINE.CONTINUATION_INDENT + LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK + ITEM'LENGTH > LINE.LENGTH then
      raise LAYOUT_ERROR;
    end if;
    if ITEM = " " and then LINE.END_PHANTOM.all = "" then
      return;
    end if;
    PUT_LINE(FILE,LINE.DATA(1..LINE.BREAK-1) & LINE.END_PHANTOM.all);
    for I in 1..LINE.INDENT + LINE.CONTINUATION_INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
    NEW_BREAK := LINE.INDENT + LINE.CONTINUATION_INDENT + 1;
    NEW_INDEX := NEW_BREAK + LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK;
    LINE.DATA(NEW_BREAK..NEW_INDEX-1) := LINE.START_PHANTOM.all &
        LINE.DATA(LINE.BREAK..LINE.INDEX-1);
    LINE.BREAK := NEW_BREAK;
    LINE.INDEX := NEW_INDEX;
  end if;
  NEW_INDEX := LINE.INDEX + ITEM'LENGTH;
  LINE.DATA(LINE.INDEX..NEW_INDEX-1) := ITEM;
  LINE.INDEX := NEW_INDEX;
  if BRK = BREAK then
```

```
    LINE.BREAK := NEW_INDEX;
  end if;
  LINE.USED_YET := TRUE;
end PRINT;

procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in STRING; BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if LINE.INDEX /= LINE.INDENT + 1 then
    PUT_LINE(FILE,LINE.DATA(1..LINE.INDEX-1));
  end if;
  for I in 1..LINE.INDENT loop
    LINE.DATA(I) := ' ';
  end loop;
  LINE.INDEX := LINE.INDENT + 1;
  LINE.BREAK := LINE.INDEX;
end PRINT_LINE;

procedure PRINT_LINE(FILE : in FILE_TYPE) is
begin
  PRINT_LINE(FILE,DEFAULT_LINE);
end PRINT_LINE;

procedure PRINT_LINE(LINE : in LINE_TYPE) is
begin
  PRINT_LINE(CURRENT_OUTPUT,LINE);
end PRINT_LINE;

procedure PRINT_LINE is
begin
  PRINT_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
```

```
end PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if LINE.USED_YET then
    NEW_LINE(FILE);
  end if;
end BLANK_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE) is
begin
  BLANK_LINE(FILE,DEFAULT_LINE);
end BLANK_LINE;

procedure BLANK_LINE(LINE : in LINE_TYPE) is
begin
  BLANK_LINE(CURRENT_OUTPUT,LINE);
end BLANK_LINE;

procedure BLANK_LINE is
begin
  BLANK_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
end BLANK_LINE;

package body INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
    S : STRING(1..NUM'WIDTH);
    L : NATURAL;
  begin
    PRINT(S,L,ITEM);
    PRINT(FILE,LINE,S(1..L),BRK);
  end PRINT;

  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
```

```
    end PRINT;

    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
    end PRINT;

    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
      S : constant STRING := NUM'IMAGE(ITEM);
      F : NATURAL := S'FIRST; -- Bug in DG Compiler -- S'FIRST /= 1 ! ! ! ! ! !
      L : NATURAL;
    begin
      if S(F) = ' ' then
        F := F + 1;
      end if;
      if TO'LENGTH < S'LAST - F + 1 then
        raise LAYOUT_ERROR;
      end if;
      L := TO'FIRST + S'LAST - F;
      TO(TO'FIRST..L) := S(F..S'LAST);
      LAST := L;
    end PRINT;

  end INTEGER_PRINT;

  package body FLOAT_PRINT is

    package NUM_IO is new FLOAT_IO(NUM);
      use NUM_IO;

    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
      S : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
      L : NATURAL;
    begin
      PRINT(S,L,ITEM);
      PRINT(FILE,LINE,S(1..L),BRK);
    end PRINT;

    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
    end PRINT;
```

```
procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in NUM;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in NUM;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
  S         : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
  EXP       : INTEGER;
  E_INDEX   : NATURAL := S'LAST - DEFAULT_EXP;
  DOT_INDEX : NATURAL := DEFAULT_FORE + 1;
  L         : NATURAL;
begin
  PUT(S,ITEM);
  EXP := INTEGER'VALUE(S(E_INDEX+1..S'LAST));
  if EXP >= 0 then
    if EXP <= DEFAULT_AFT-1 then
      S(DOT_INDEX..DOT_INDEX+EXP-1) := S(DOT_INDEX+1..DOT_INDEX+EXP);
      S(DOT_INDEX+EXP) := '.';
      for I in E_INDEX..S'LAST loop
        S(I) := ' ';
      end loop;
    end if;
  else -- EXP < 0
    if EXP >= - ( DEFAULT_EXP + 1 ) then
      S(DEFAULT_EXP+2..S'LAST) := S(1..S'LAST-DEFAULT_EXP-1);
      for I in 1..DEFAULT_EXP+1 loop
        S(I) := ' ';
      end loop;
      E_INDEX := S'LAST + 1;
      DOT_INDEX := DOT_INDEX + DEFAULT_EXP + 1;
      L := DOT_INDEX+EXP;
      for I in reverse L+1..DOT_INDEX loop
        case S(I-1) is
          when ' '    => S(I) := '0';
          when '-'    => S(I-2) := '-'; S(I) := '0';
          when others => S(I) := S(I-1);
        end case;
      end loop;
      S(L) := '.';
      case S(L-1) is
        when ' '    => S(L-1) := '0';
```

```
          when '-'    => S(L-2) := '-'; S(L-1) := '0';
          when others => null;
        end case;
      end if;
    end if;
    for I in reverse 1..E_INDEX-1 loop
      exit when S(I) /= '0' or else S(I-1) = '.';
      S(I) := ' ';
    end loop;
    L := TO'FIRST - 1;
    for I in S'RANGE loop
      if S(I) /= ' ' then
        L := L + 1;
        TO(L) := S(I);
      end if;
    end loop;
    LAST := L;
  exception
    when CONSTRAINT_ERROR =>
      raise LAYOUT_ERROR;
  end PRINT;

 end FLOAT_PRINT;

end TEXT_PRINT;
```

3.11.7 package lexs.ada

```
with SYSTEM;
package LEXICAL_ANALYZER is

   -- Description:
   --
   --    The Lexical Analyzer combines two major functions into one integrated
   -- package:  token input and diagnostic reporting.  The Lexical Analyzer
   -- provides facilities to provide lexical tokens to the caller for a
   -- specific input file.  Diagnostic reporting is supported by providing
   -- subprograms that support the reporting of six different types of
   -- diagnostic messages including:  syntax errors, semantic errors, fatal
   -- errors, system errors, warnings and notes.

   ----------------------------------------------------------------------------

   -- The following types and objects provide configuration information to the
   -- Lexical Analyzer.  These types and object declarations may be dependent
   -- on the host Ada compiler.

   MAXIMUM_INPUT_LINE_LENGTH : constant := 132;
      -- The Lexical Analyzer reads each line from the input file into a string
```

```
   -- buffer.  This constant is used to determine the size of the string
   -- buffer.

MAXIMUM_INPUT_LINES : constant := SYSTEM.MAX_INT;
   -- The Lexical Analyzer assigns line numbers to each input line read to
   -- facilitate error reporting.

IGNORE_PRAGMAS : BOOLEAN := TRUE;
   -- The Lexical Analyzer will not return a pragma as a token if this
   -- variable is set to true.  Instead, the Lexical Analyzer will parse
   -- the pragma internally (for syntax only) and skip over all the tokens
   -- associated with the pragma.  Since pragmas can occur almost anywhere
   -- in Ada, this capability will free the user of the Lexical Analyzer
   -- from the chore of always checking for the presence of a pragma in
   -- parsing the input stream.

MAXIMUM_NUMBER_OF_ERRORS : INTEGER := 100;
   -- This integer indicates the maximum number of errors that will be
   -- reported.  If there are more than this number of errors, the Lexical
   -- Analyzer will generate a fatal error as soon as the limit is exceeded.
   -- The fatal error should cause the Application Scanner to abort
   -- processing.

LINES_PER_PAGE_FOR_ERROR_LISTING : INTEGER := 55;
   -- Controls the number of lines written to the error listing file before
   -- a new page is issued.  This value does not include the heading lines
   -- generated at the top of each page.

COLUMNS_PER_LINE_FOR_ERROR_LISTING : INTEGER := 80;
   -- Controls the maximum length of each line written to the error listing
   -- file.  Source lines which are being displayed to the error listing
   -- will be truncated at this length.

MESSAGE_WRAP_LENGTH : INTEGER := 80;
   -- Controls the maximum length of a diagnostic message in the error
   -- listing file.  Diagnostic messages that exceed this length are word
   -- wrapped to the next line.

INDENT_FOR_MESSAGE_WRAP : INTEGER := 10;
   -- Controls the indentation used for diagnostic messages that are wrapped.
   -- The indentation is used for subsequent lines of the diagnostic
   -- message.

DISPLAY_ERRORS_IMMEDIATELY : BOOLEAN := FALSE;
   -- The Lexical Analyzer will report the error immediately to the error
   -- file if this variable is true.  This error reporting is in addition
   -- to the reporting that is done at the end of processing by the
   -- subprogram PRODUCE_ERROR_LISTING.  This capability is useful if the
   -- Application Scanner aborts before it has the chance to call
```

```
    -- PRODUCE_ERROR_LISTING.

----------------------------------------------------------------------------

-- A delimiter is either one of the following special characters (in the
-- basic character set)
--
--     & ' ( ) * + , - . / : ; < = > |
--
-- or one of the following compound delimiters each composed of two adjacent
-- special characters
--
--     =>  ..  **  :=  /=  >=  <=  <<  >>  <>
--
-- Each of the special characters listed for single character delimiters is
-- a single delimiter except if this character is used as a character of a
-- compound delimiter, or as a character of a comment, string literal,
-- character literal, or numeric literal.

type DELIMITER_KIND is
   (
      AMPERSAND,                APOSTROPHE,              LEFT_PARENTHESIS,
      RIGHT_PARENTHESIS,        STAR,                    PLUS,
      COMMA,                    HYPHEN,                  DOT,
      SLASH,                    COLON,                   SEMICOLON,
      LESS_THAN,                EQUAL,                   GREATER_THAN,
      VERTICAL_BAR,             ARROW,                   DOUBLE_DOT,
      DOUBLE_STAR,              ASSIGNMENT,              INEQUALITY,
      GREATER_THAN_OR_EQUAL,    LESS_THAN_OR_EQUAL,      LEFT_LABEL_BRACKET,
      RIGHT_LABEL_BRACKET,      BOX
   );

----------------------------------------------------------------------------

-- The Ada reserved words are reserved for special significance in the
-- language.  A reserved word must not be used as a declared identifier.

type RESERVED_WORD_KIND is
   (
      R_ABORT,       R_ABS,          R_ACCEPT,      R_ACCESS,      R_ALL,
      R_AND,         R_ARRAY,        R_AT,          R_BEGIN,       R_BODY,
      R_CASE,        R_CONSTANT,     R_DECLARE,     R_DELAY,       R_DELTA,
      R_DIGITS,      R_DO,           R_ELSE,        R_ELSIF,       R_END,
      R_ENTRY,       R_EXCEPTION,    R_EXIT,        R_FOR,         R_FUNCTION,
      R_GENERIC,     R_GOTO,         R_IF,          R_IN,          R_IS,
      R_LIMITED,     R_LOOP,         R_MOD,         R_NEW,         R_NOT,
      R_NULL,        R_OF,           R_OR,          R_OTHERS,      R_OUT,
```

```
      R_PACKAGE,     R_PRAGMA,     R_PRIVATE,     R_PROCEDURE,  R_RAISE,
      R_RANGE,       R_RECORD,     R_REM,         R_RENAMES,    R_RETURN,
      R_REVERSE,     R_SELECT,     R_SEPARATE,    R_SUBTYPE,    R_TASK,
      R_TERMINATE,   R_THEN,       R_TYPE,        R_USE,        R_WHEN,
      R_WHILE,       R_WITH,       R_XOR
   );

-----------------------------------------------------------------------------

-- The Lexical Analyzer determines the kind of each token and remembers the
-- line number and the position in the line of the first lexical element of
-- the token.

type TOKEN_KIND is
   (
      IDENTIFIER, NUMERIC_LITERAL,  CHARACTER_LITERAL,   STRING_LITERAL,
      DELIMITER,  RESERVED_WORD,    END_OF_FILE
   );

subtype  SOURCE_POSITION   is NATURAL range 0 .. MAXIMUM_INPUT_LINE_LENGTH + 1
type     SOURCE_LINE       is range 0 .. MAXIMUM_INPUT_LINES;
type     STRING_ACCESS     is access STRING;

-----------------------------------------------------------------------------

-- The following type defines the contents of a Lexical Token.

type LEXICAL_TOKEN_RECORD (KIND : TOKEN_KIND) is
   record
      LINE  : SOURCE_LINE;
      START : SOURCE_POSITION;

      case KIND is
         when IDENTIFIER =>
            ID : STRING_ACCESS;
               -- Points to the upper-cased image of the identifier with
               -- no padding (e.g., ID.all'LENGTH = number of characters
               -- in the identifier.  Use the FIRST and LAST attributes
               -- to access individual characters in the string.

         when NUMERIC_LITERAL =>
            IMAGE : STRING_ACCESS;
               -- Points to the image of the numeric literal as
               -- it appeared in the source.
               -- To get the value, use the VALUE attribute.

         when CHARACTER_LITERAL =>
            CHARACTER_VALUE : CHARACTER;
```

```
         when STRING_LITERAL =>
            STRING_IMAGE : STRING_ACCESS;
               -- Points to the image of the string literal.  The image does
               -- not include the surrounding quotation marks.  Also, each
               -- doubled quotation character in the input string literal is
               -- translated into a single quotation character in this image.
               -- The case of the alphabetic characters is the same as the
               -- input string literal (i.e., the characters in the string
               -- have not been upper-cased.

         when DELIMITER =>
            DELIMITER : DELIMITER_KIND;

         when RESERVED_WORD =>
            RESERVED_WORD : RESERVED_WORD_KIND;

         when END_OF_FILE =>
            null;

      end case;
   end record;

   type LEXICAL_TOKEN is access LEXICAL_TOKEN_RECORD;

------------------------------------------------------------------------------

-- The following subprograms are provided to open and close the token
-- input file.  Currently, only one input file may be open at a time and
-- all tokens are retrieved from the current input file.  A fatal error is
-- generated if the file cannot be opened or closed.  If the filename opened
-- is STANDARD_INPUT, then the STANDARD_INPUT file of TEXT_IO is used
-- instead of opening a new file.

procedure OPEN_TOKEN_STREAM
   (UNIT_FILENAME    : in STRING;
    LISTING_FILENAME : in STRING := "");

procedure CLOSE_TOKEN_STREAM;

------------------------------------------------------------------------------

-- The following subprograms are provided to retrieve tokens from the
-- current input file.  The Lexical Analyzer supports an infinite look-ahead
-- of tokens without changing the next token (i.e., the first look-ahead
-- token).  The subprograms implement this look-ahead by maintaining two
-- separate lexical pointers into the current input file:  the current
-- lexical pointer and the next-look-ahead lexical pointer.
--
--    NEXT_TOKEN returns the next token in the current input file.  This
```

```
--        action advances the current lexical pointer.
--
--     EAT_NEXT_TOKEN is similar to NEXT_TOKEN except the token is not
--        returned to the caller.  This action advances the current lexical
--        pointer.
--
--     FIRST_LOOK_AHEAD_TOKEN returns the next token in the current input
--        file but does not advance the current lexical pointer but does
--        advance the "next look-ahead" lexical pointer.  That is, the token
--        that would subsequently be returned by NEXT_TOKEN is the same as
--        the token returned by this subprogram.
--
--     SET_LOOK_AHEAD positions the "next look-ahead" lexical pointer to
--        the current lexical pointer.
--
--     NEXT_LOOK_AHEAD_TOKEN returns the next look-ahead token in the current
--        input file as indicated by the "next look-ahead" lexical pointer.
--        This action advances the "next look-ahead" lexical pointer, but
--        does not affect the current lexical pointer.

function NEXT_TOKEN              return LEXICAL_TOKEN;
function FIRST_LOOK_AHEAD_TOKEN  return LEXICAL_TOKEN;
function NEXT_LOOK_AHEAD_TOKEN   return LEXICAL_TOKEN;

procedure SET_LOOK_AHEAD;
procedure EAT_NEXT_TOKEN;

-----------------------------------------------------------------------------

-- The following subprograms allow tokens to be skipped temporarily in the
-- current token stream and restored later so that they may be processed
-- just as if they had actually occurred in the current file at the later
-- position.  The only known use for this facility is for the processing of
-- a query expression where the select items must be skipped temporarily so
-- that the from clause may be processed first.  To handle this case, each
-- token in the select items list would be skipped by a call to
-- SKIP_TOKEN_FOR_NOW.  Later, after processing the from clause, the
-- subprogram RESTORE_SKIPPED_TOKENS would be called after which the select
-- items list could be processed just as though it had followed the from
-- clause rather than preceding it.
--
--  KIP_TOKEN_FOR_NOW has the same affect on token processing as the
-- subprogram EAT_NEXT_TOKEN except that the token skipped is remembered by
-- the Lexical Analyzer.

procedure SKIP_TOKEN_FOR_NOW;
procedure RESTORE_SKIPPED_TOKENS;

-----------------------------------------------------------------------------
```

```
-- The following subprograms and exceptions provide diagnostic support.

SYNTAX_ERROR : exception;
FATAL_ERROR  : exception;
SYSTEM_ERROR : exception;

-- SYNTAX_ERROR reports the error message and raises the exception SYNTAX_ERROR.
-- May cause a FATAL error to be generated if the maximum number of error
-- messages have already been issued.
--
procedure REPORT_SYNTAX_ERROR (TOKEN : in LEXICAL_TOKEN; MESSAGE : in STRING);

-- REPORT_DDL_ERROR reports the errors from the ddl, since they will not
-- be synced with the application scanner's source it needs a separate
-- routine
procedure REPORT_DDL_ERROR (MESSAGE : in STRING);

-- SEMANTIC_ERROR reports the error message but does not raise any
-- exceptions.  May cases a FATAL error to be generated if the maximum
-- number of error messages have already been issued.
--
procedure REPORT_SEMANTIC_ERROR (TOKEN : in LEXICAL_TOKEN; MESSAGE : in STRING);

-- FATAL_ERROR reports the error message and raises the exception FATAL_ERROR.
-- No subsequent calls to the Lexical Analyzer should be made to issue
-- other diagnostic messages or retrieve tokens from the current input file.
--
procedure REPORT_FATAL_ERROR (TOKEN : in LEXICAL_TOKEN; MESSAGE : in STRING);
procedure REPORT_FATAL_ERROR (MESSAGE : in STRING);

-- SYSTEM_ERROR reports the error message and raises the exception
-- SYSTEM_ERROR.  This should be used when an internal error is detected in
-- processing.
--
procedure REPORT_SYSTEM_ERROR (TOKEN   : in LEXICAL_TOKEN; MESSAGE : in STRING);
procedure REPORT_SYSTEM_ERROR (MESSAGE : in STRING);

-- WARNING reports a warning message to the error file but does not raise
-- any exceptions.
--
procedure REPORT_WARNING (TOKEN : in LEXICAL_TOKEN; MESSAGE : in STRING);

-- NOTE reports a note message but does not raise any exceptions.  This can
-- be used to provide explanation.
--
procedure REPORT_NOTE (TOKEN : in LEXICAL_TOKEN; MESSAGE : in STRING);

-- SEVERE_ERRORS returns the number of severe errors (syntax, semantic,
-- system, or fatal errors) generated thus far.  Note that warnings and
```

```
   -- notes are not included in this count.
   --
   function SEVERE_ERRORS return INTEGER;

   -- PRODUCE_ERROR_LISTING writes the error listing to the error file.  Raises
   -- a system error if the current input file is not closed.  This should
   -- only be called after processing has been complete.
   --
   procedure PRODUCE_ERROR_LISTING;

   -- For debugging purposes:
   procedure PRINT_TOKEN (TOKEN : in LEXICAL_TOKEN);

end LEXICAL_ANALYZER;
```

3.11.8 package lexb.ada

```
with TEXT_IO, UNCHECKED_DEALLOCATION;
package body LEXICAL_ANALYZER is

   type TOKEN_LIST_ENTRY_RECORD;
   type TOKEN_LIST_ENTRY is access TOKEN_LIST_ENTRY_RECORD;

   type TOKEN_LIST_ENTRY_RECORD is
      record
         TOKEN : LEXICAL_TOKEN;
         NEXT  : TOKEN_LIST_ENTRY;
      end record;

   type MESSAGE_KIND is
      (SYNTAX, SEMANTIC, FATAL, SYSTEM, WARNING, NOTE);

   type MESSAGE_LIST_ENTRY_RECORD;
   type MESSAGE_LIST_ENTRY is access MESSAGE_LIST_ENTRY_RECORD;

   type MESSAGE_LIST_ENTRY_RECORD is
      record
         LINE     : SOURCE_LINE := 0;
         START    : SOURCE_POSITION := 0;
         KIND     : MESSAGE_KIND;
         MESSAGE  : STRING_ACCESS;
         NEXT     : MESSAGE_LIST_ENTRY;
      end record;

   type ERROR_COUNT_ARRAY is array (MESSAGE_KIND) of NATURAL;

   type FILE;
   type FILE_LIST is access FILE;

   type FILE is
```

```
   record
      NAME               : STRING_ACCESS;
      STREAM             : TEXT_IO.FILE_TYPE;
      IS_OPEN            : BOOLEAN := FALSE;
      EOF                : BOOLEAN := FALSE;
      BUFFER             : STRING (1..MAXIMUM_INPUT_LINE_LENGTH);
      LAST               : SOURCE_POSITION := 0;
      LINE               : SOURCE_LINE := 0;
      NEXT               : SOURCE_POSITION := 1;
      LOOK_AHEAD_TOKENS  : TOKEN_LIST_ENTRY;
      LOOK_AHEAD_PTR     : INTEGER := 0;
      SKIPPED_TOKENS     : TOKEN_LIST_ENTRY;
      MESSAGE_LIST       : MESSAGE_LIST_ENTRY;
      ERROR_COUNT        : ERROR_COUNT_ARRAY := (others => 0);
      SHADOW_FILE        : TEXT_IO.FILE_TYPE;
      SHADOW_FILE_OPEN   : BOOLEAN := FALSE;
      ERROR_FILENAME     : STRING_ACCESS;
      ERROR_FILE         : TEXT_IO.FILE_TYPE;
      ERROR_FILE_PAGE    : NATURAL := 1;
      ERROR_FILE_LINE    : NATURAL := 0;
      USE_STANDARD_INPUT  : BOOLEAN := FALSE;
      USE_STANDARD_OUTPUT : BOOLEAN := FALSE;
      PREVIOUS_FILE      : FILE_LIST;
   end record;

CURRENT_FILE          : FILE_LIST;
EOF_TOKEN             : LEXICAL_TOKEN;

-- The following constants are used in error recovery.  Their values are
-- unimportant but they must be unique.
DELETED_CHAR                 : constant CHARACTER := ASCII.NUL;
INSERTED_ZERO_AFTER_DOT      : constant CHARACTER := ASCII.STX;
INSERTED_ZERO_AFTER_E        : constant CHARACTER := ASCII.EOT;
INSERTED_ZERO_AFTER_MINUS    : constant CHARACTER := ASCII.ACK;
INSERTED_ZERO_AFTER_PLUS     : constant CHARACTER := ASCII.BS;
INSERTED_ZERO_AFTER_ONE      : constant CHARACTER := ASCII.LF;
INSERTED_ZERO_AFTER_SHARP    : constant CHARACTER := ASCII.FF;
INSERTED_ZERO_AFTER_COLON    : constant CHARACTER := ASCII.SO;

-- Some forward declarations for issuing diagnostics where there is not token.
-- Diagnostic will be issued at current position in file.

procedure REPORT_SYNTAX_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING);

procedure REPORT_SEMANTIC_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING);

procedure REPORT_SYSTEM_ERROR
```

```
   (COL : in SOURCE_POSITION; MESSAGE : in STRING);

procedure REPORT_FATAL_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING);

procedure FREE is new UNCHECKED_DEALLOCATION (STRING, STRING_ACCESS);
procedure FREE is new UNCHECKED_DEALLOCATION
   (TOKEN_LIST_ENTRY_RECORD, TOKEN_LIST_ENTRY);

procedure OPEN_TOKEN_STREAM
   (UNIT_FILENAME    : in STRING;
    LISTING_FILENAME : in STRING := "") is
   NEW_FILE : FILE_LIST := new FILE;
begin
   NEW_FILE.PREVIOUS_FILE := CURRENT_FILE;
   NEW_FILE.USE_STANDARD_INPUT := (UNIT_FILENAME = "STANDARD_INPUT");
   if not NEW_FILE.USE_STANDARD_INPUT then
      TEXT_IO.OPEN (NEW_FILE.STREAM, TEXT_IO.IN_FILE, UNIT_FILENAME);
   end if;
   NEW_FILE.NAME := new STRING(1..UNIT_FILENAME'LENGTH);
   NEW_FILE.NAME.all := UNIT_FILENAME;
   NEW_FILE.IS_OPEN := TRUE;
   NEW_FILE.USE_STANDARD_OUTPUT := (LISTING_FILENAME = "STANDARD_OUTPUT");
   NEW_FILE.ERROR_FILENAME := new STRING (1..LISTING_FILENAME'LENGTH);
   NEW_FILE.ERROR_FILENAME.all := LISTING_FILENAME;
   if LISTING_FILENAME /= "" then
      -- Create the shadow file.
      TEXT_IO.CREATE (NEW_FILE.SHADOW_FILE);
      NEW_FILE.SHADOW_FILE_OPEN := TRUE;
   end if;
   CURRENT_FILE := NEW_FILE;
exception
   when others =>
      REPORT_FATAL_ERROR ("Unable to open file: " & UNIT_FILENAME);
end OPEN_TOKEN_STREAM;

procedure CLOSE_TOKEN_STREAM is
begin
   if CURRENT_FILE /= null and then CURRENT_FILE.IS_OPEN then
      if not CURRENT_FILE.USE_STANDARD_INPUT then
         TEXT_IO.CLOSE (CURRENT_FILE.STREAM);
      end if;
      CURRENT_FILE.IS_OPEN := FALSE;
   end if;
exception
   when others =>
      if CURRENT_FILE /= null then
         CURRENT_FILE.IS_OPEN := FALSE;
      end if;
```

```
        REPORT_FATAL_ERROR
            ("Unable to close file: " & CURRENT_FILE.NAME.all);
    end CLOSE_TOKEN_STREAM;

    procedure NEXT_LINE is
    begin
        if not CURRENT_FILE.EOF then
            loop
                CURRENT_FILE.LINE := CURRENT_FILE.LINE + 1;
                TEXT_IO.GET_LINE
                    (CURRENT_FILE.STREAM, CURRENT_FILE.BUFFER, CURRENT_FILE.LAST);
                if CURRENT_FILE.ERROR_FILENAME.all /= "" then
                    TEXT_IO.PUT_LINE
                        (CURRENT_FILE.SHADOW_FILE,
                         CURRENT_FILE.BUFFER(1..CURRENT_FILE.LAST));
                end if;
            exit when CURRENT_FILE.LAST > 0;
            end loop;
            CURRENT_FILE.NEXT := 1;
        end if;
    exception
        when TEXT_IO.END_ERROR =>
            CURRENT_FILE.EOF := TRUE;
    end NEXT_LINE;

    procedure ADVANCE_TO_NEXT_TOKEN is

        function WHITESPACE (C : in CHARACTER) return BOOLEAN is
        begin
            case C is
                when ' ' | ASCII.HT | ASCII.VT | ASCII.CR | ASCII.LF | ASCII.FF =>
                    return TRUE;
                when others =>
                    return FALSE;
            end case;
        end WHITESPACE;

    begin
        while not CURRENT_FILE.EOF loop
            if CURRENT_FILE.NEXT > CURRENT_FILE.LAST then
                NEXT_LINE;
            elsif CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT) = '-' and then
                CURRENT_FILE.NEXT < CURRENT_FILE.LAST and then
                CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT+1) = '-' then
                NEXT_LINE;
            elsif not WHITESPACE(CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT)) then
                exit;
            else
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
```

```
      end if;
   end loop;
end ADVANCE_TO_NEXT_TOKEN;

function GET_EOF_TOKEN return LEXICAL_TOKEN is
begin
   if EOF_TOKEN = null then
      EOF_TOKEN := new LEXICAL_TOKEN_RECORD (END_OF_FILE);
   end if;
   EOF_TOKEN.LINE := CURRENT_FILE.LINE;
   EOF_TOKEN.START := CURRENT_FILE.NEXT;
   return EOF_TOKEN;
end GET_EOF_TOKEN;

function START_OF_CHARACTER_LITERAL return BOOLEAN is
begin
   -- This is the start of a character literal if the first and third
   -- characters are apostrophes.
   if CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT) = ''' and then
      CURRENT_FILE.NEXT + 2 <= CURRENT_FILE.LAST and then
      CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT + 2) = ''' then
      return TRUE;
   else
      return FALSE;
   end if;
end START_OF_CHARACTER_LITERAL;

function START_OF_STRING_LITERAL return BOOLEAN is
   C : CHARACTER renames CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT);
begin
   -- This is the start of a string literal if the first character is a
   -- quotation mark or a percent.
   return C = '"' or C = '%';
end START_OF_STRING_LITERAL;

function START_OF_NUMERIC_LITERAL return BOOLEAN is
   C : CHARACTER renames CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT);
begin
   -- This is the start of a numeric literal if the first character is a
   -- digit.
   return C in '0' .. '9';
end START_OF_NUMERIC_LITERAL;

function START_OF_IDENTIFIER return BOOLEAN is
   C : CHARACTER renames CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT);
begin
   -- This is the start of an identifier if the first character is
   -- alphabetic.
   return C in 'A' .. 'Z' or C in 'a' .. 'z';
```

```
end START_OF_IDENTIFIER;

function START_OF_DELIMITER return BOOLEAN is
   C : CHARACTER renames CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT);
begin
   -- This is the start of a delimiter if the first character is
   -- either:
   --
   --     & ' ( ) * + , - . / : ; < = > | !
   case C is
      when '&' | ''' | '(' | ')' | '*' | '+' | ',' | '-' | '.' |
           '/' | ':' | ';' | '<' | '=' | '>' | '|' | '!' =>
         return TRUE;
      when others =>
         return FALSE;
   end case;
end START_OF_DELIMITER;

function GET_CHARACTER_LITERAL return LEXICAL_TOKEN is
   TOKEN : LEXICAL_TOKEN;
begin
   -- The following check is just to validate the assumption that the next
   -- token is actually a character literal.  This should already have been
   -- validated before the call to this subprogram, but we do it here to
   -- make sure.
   if CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT) /= ''' or else
      CURRENT_FILE.NEXT + 2 > CURRENT_FILE.LAST or else
      CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT + 2) /= ''' then
      REPORT_SYSTEM_ERROR
         (CURRENT_FILE.NEXT,
          "Expecting valid character literal in GET_CHARACTER_LITERAL.");
   end if;
   -- Make a new character literal token.
   TOKEN := new LEXICAL_TOKEN_RECORD (CHARACTER_LITERAL);
   TOKEN.LINE := CURRENT_FILE.LINE;
   TOKEN.START := CURRENT_FILE.NEXT;
   TOKEN.CHARACTER_VALUE := CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT + 1);
   -- Advance the NEXT pointer by the length of the character literal.
   CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 3;
   return TOKEN;
end GET_CHARACTER_LITERAL;

function GET_STRING_LITERAL return LEXICAL_TOKEN is
   TOKEN           : LEXICAL_TOKEN;
   DELIMITER       : CHARACTER := CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT);
   START_OF_STRING : SOURCE_POSITION := CURRENT_FILE.NEXT + 1;
   STRING_LENGTH   : NATURAL := 0;
   INDEX           : SOURCE_POSITION;
begin
```

```
      -- A string literal starts with either an quotation mark or alternatively
      -- a percent.  Verify this fact.
      if DELIMITER /= '"' and DELIMITER /= '%' then
         REPORT_SYSTEM_ERROR
            (CURRENT_FILE.NEXT,
             "Invalid string literal delimiter encountered.");
      end if;
      loop
         CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         if CURRENT_FILE.NEXT > CURRENT_FILE.LAST then
            REPORT_SEMANTIC_ERROR
               (START_OF_STRING-1,
                "Unterminated character string");
       exit;  -- For error recovery.
         elsif CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT) = DELIMITER then
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
            -- exit if this is not a doubled delimiter.
       exit when CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
          CURRENT_FILE.BUFFER (CURRENT_FILE.NEXT) /= DELIMITER;
         end if;
         STRING_LENGTH := STRING_LENGTH + 1;
      end loop;
      -- Now, CURRENT_FILE.NEXT points to the character past the ending
      -- delimiter for the string literal.  STRING_LENGTH indicates the
      -- number of characters in the string (counting a doubled delimiter as
      -- 1 character).  START_OF_STRING points to the first character (if
      -- any) of the string literal, just past the first delimiter.

      TOKEN := new LEXICAL_TOKEN_RECORD (STRING_LITERAL);
      TOKEN.LINE := CURRENT_FILE.LINE;
      TOKEN.START := START_OF_STRING - 1;
      TOKEN.STRING_IMAGE := new STRING (1..STRING_LENGTH);
      -- Copy the string literal to the token with only copying one delimiter
      -- for each doubled delimiter found.
      INDEX := START_OF_STRING;
      for I in NATURAL range 1..STRING_LENGTH loop
         if CURRENT_FILE.BUFFER(INDEX) = DELIMITER then
            INDEX := INDEX + 1;
         end if;
         TOKEN.STRING_IMAGE.all(I) := CURRENT_FILE.BUFFER(INDEX);
         INDEX := INDEX + 1;
      end loop;
      return TOKEN;
   end GET_STRING_LITERAL;


   function GET_NUMERIC_LITERAL return LEXICAL_TOKEN is
      subtype BASE_TYPE is INTEGER range 2..16;
      BASE                  : BASE_TYPE;
```

```
TOKEN                 : LEXICAL_TOKEN;
START_OF_NUMERIC      : SOURCE_POSITION := CURRENT_FILE.NEXT;
DELIMITER             : CHARACTER;
SAVE_ERROR_POSITION : SOURCE_POSITION;
STRING_LENGTH         : NATURAL;
INDEX                 : NATURAL;
DIGIT_SEEN            : BOOLEAN;
NONZERO_DIGIT_SEEN    : BOOLEAN;

procedure SCAN_TO_END_OF_DECIMAL_INTEGER is
begin
   -- integer ::= digit {[underline] digit}
   if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) in '0' .. '9' then
      CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
   else
      -- This should never happen since it should be verified before
      -- calling this subprogram.
      REPORT_SYSTEM_ERROR
         (CURRENT_FILE.NEXT,
          "Expecting numeric value.");
   end if;
   loop
      -- Skip optional underline.
      if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' then
         CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         -- A digit must follow an underline.  Verify.
         if CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) not in '0' .. '9' then
            if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
               CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' then
               REPORT_SEMANTIC_ERROR
                  (CURRENT_FILE.NEXT,
                   "Illegal double underline; deleted ""_""");
               CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
               CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
            else
               REPORT_SEMANTIC_ERROR
                  (CURRENT_FILE.NEXT-1,
                   "Deleted illegal trailing underline");
               CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-1) := DELETED_CHAR;
            end if;
         end if;
      end if;
      exit when CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) not in '0' .. '9';

      CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
```

```
   end loop;
end SCAN_TO_END_OF_DECIMAL_INTEGER;

procedure SCAN_TO_END_OF_BASED_INTEGER
   (BASE : in BASE_TYPE;
    DEL  : in CHARACTER) is

   function VALUE_OF_HEXIDECIMAL_CHARACTER
      (HEX_CHARACTER : in CHARACTER) return INTEGER is
   begin
      case HEX_CHARACTER is
         when '0' => return 0;
         when '1' => return 1;
         when '2' => return 2;
         when '3' => return 3;
         when '4' => return 4;
         when '5' => return 5;
         when '6' => return 6;
         when '7' => return 7;
         when '8' => return 8;
         when '9' => return 9;
         when 'A' | 'a' => return 10;
         when 'B' | 'b' => return 11;
         when 'C' | 'c' => return 12;
         when 'D' | 'd' => return 13;
         when 'E' | 'e' => return 14;
         when 'F' | 'f' => return 15;
         when others =>
            REPORT_SYSTEM_ERROR
              (CURRENT_FILE.NEXT,
               "Expecting hex character.");
      end case;
   end VALUE_OF_HEXIDECIMAL_CHARACTER;

   function IS_HEXIDECIMAL_CHARACTER
      (C : in CHARACTER) return BOOLEAN is
   begin
      if C in '0'..'9' or else
         C in 'a'..'f' or else
         C in 'A'..'F' then
         return TRUE;
      else
         return FALSE;
      end if;
   end IS_HEXIDECIMAL_CHARACTER;

begin
   -- based_integer ::= extended_digit {[underline] extended_digit
   -- extended_digit ::= digit | letter
```

```
DIGIT_SEEN := FALSE;

if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
   IS_HEXIDECIMAL_CHARACTER (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) then
   DIGIT_SEEN := TRUE;
   if VALUE_OF_HEXIDECIMAL_CHARACTER
      (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) >= BASE then
      REPORT_SEMANTIC_ERROR
         (CURRENT_FILE.NEXT,
          "Illegal digit for base " & BASE_TYPE'IMAGE(BASE) &
          "; ""0"" assumed");
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := '0';
   end if;
   CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
else
   -- Unexpected characters.  For recovery, just skip over these.
   SAVE_ERROR_POSITION := CURRENT_FILE.NEXT;
   while CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
      not IS_HEXIDECIMAL_CHARACTER
         (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) and then
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) /= DEL loop
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
      CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
   end loop;
   if CURRENT_FILE.NEXT > CURRENT_FILE.LAST then
      REPORT_SEMANTIC_ERROR
         (SAVE_ERROR_POSITION,
          "Expecting based digit or '" & DEL & "'; remainder of line" &
          " ignored");
      if SAVE_ERROR_POSITION > CURRENT_FILE.LAST then
         -- To recover, extend line and put a delimiter in its place.
         CURRENT_FILE.LAST := CURRENT_FILE.LAST + 1;
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DEL;
      else
         -- To recover, put delimiter at end of line.
         CURRENT_FILE.NEXT := CURRENT_FILE.LAST;
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DEL;
      end if;
   else
      REPORT_SEMANTIC_ERROR
         (SAVE_ERROR_POSITION,
          "Expecting based digit or '" & DEL &
          "'; unexpected character(s) ignored");
   end if;
end if;
loop
   -- Skip optional underline.
   if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' then
```

```
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
            -- A hexidecimal digit must follow an underline.  Verify.
            if CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
               (not IS_HEXIDECIMAL_CHARACTER
                  (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT))) then
               if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' then
                  REPORT_SEMANTIC_ERROR
                     (CURRENT_FILE.NEXT,
                      "Illegal double underline; deleted ""_""");
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
                  CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               else
                  REPORT_SEMANTIC_ERROR
                     (CURRENT_FILE.NEXT-1,
                      "Deleted illegal trailing underline");
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-1) := DELETED_CHAR;
               end if;
            end if;
         end if;
      exit when CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
         not IS_HEXIDECIMAL_CHARACTER
            (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT));

         DIGIT_SEEN := TRUE;

         if VALUE_OF_HEXIDECIMAL_CHARACTER
            (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) >= BASE then
            REPORT_SEMANTIC_ERROR
               (CURRENT_FILE.NEXT,
                "Illegal digit for base " & BASE_TYPE'IMAGE(BASE) &
                "; ""0"" assumed");
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := '0';
         end if;

         CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
      end loop;
   end SCAN_TO_END_OF_BASED_INTEGER;

begin
   -- There are two classes of numeric literals:  real literals and integer
   -- literals.  A real literal is a numeric literal that includes a point;
   -- an integer literal is a numeric literal without a point.
   --
   --    numeric_literal ::= decimal_literal | based_literal
   --
   -- A decimal literal is a numeric literal expressed in the conventional
   -- decimal notation (that is, the base is implicitly ten).
   --
```

```
--    decimal_literal ::= integer [.integer] [exponent]
--    integer := digit {[underline] digit}
--    exponent ::= E [+] integer | E - integer
--
-- An underline character inserted between adjacent digits of a decimal
-- literal does not affect the value of this numeric literal.  The letter
-- E of the exponent, if any can be written either in lower case or in
-- upper case, with the same meaning.  The base and the exponent, if any,
-- are in decimal notation.
--
-- An exponent for an integer literal must not have a minus sign.
--
-- A based literal is a numeric literal expressed in a form that specifies
-- the base explicitly.  The base must be at least two and at most
-- sixteen.
--
--    based_literal  ::= base # based_integer [.based_integer] # [exponent]
--    base           ::= integer
--    based_integer  ::= extended_digit {[undeline] extended_digit}
--    extended_digit ::= digit | letter
--
-- The only letters allowed as extended digits are the letters A through
-- F for the digits ten through fifteen.  A letter in a based literal
-- (either an extended digit or the letter E of an exponent) can be
-- written either in lower case or in upper case, with the same meaning.
--
-- The conventional meaning of based notation is asssumed;  in particular
-- the value of each extended digit of a based literal must be less than
-- the base.

-- First, all numeric literals start with an integer portion.
-- Get this integer.
SCAN_TO_END_OF_DECIMAL_INTEGER;
if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
   (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '#' or
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = ':') then
   -- This is a based literal.
   DELIMITER := CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT);
   CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
   begin
      BASE := BASE_TYPE'VALUE
         (CURRENT_FILE.BUFFER (START_OF_NUMERIC..CURRENT_FILE.NEXT-2));
   exception
      when others =>
         REPORT_SEMANTIC_ERROR
            (CURRENT_FILE.NEXT-2,
             "Base must be between 2 and 16; base 10 assumed");
         -- Fix up base so equals 10.
         if CURRENT_FILE.NEXT-2 > START_OF_NUMERIC then
```

```
            for I in START_OF_NUMERIC..CURRENT_FILE.NEXT-4 loop
               CURRENT_FILE.BUFFER(I) := DELETED_CHAR;
            end loop;
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-3) := '1';
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-2) := '0';
         else
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-2) :=
               INSERTED_ZERO_AFTER_ONE;
         end if;
         BASE := 10;  -- just to continue.
   end;
   if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
      CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = DELIMITER then
      REPORT_SEMANTIC_ERROR
         (CURRENT_FILE.NEXT,
          "Missing based number; ""0"" assumed");
      if DELIMITER = '#' then
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-1) :=
            INSERTED_ZERO_AFTER_SHARP;
      else
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-1)  :=
            INSERTED_ZERO_AFTER_COLON;
      end if;
      -- For complete recovery, really need to check if there is a
      -- a based number after it has been parsed (because there may be
      -- any number of deleted characters in this number (e.g., consider
      -- 10#__#);
   else
      SCAN_TO_END_OF_BASED_INTEGER (BASE, DELIMITER);
      -- Skip optional part.
      if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
         CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '.' then
         CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = DELIMITER then
            REPORT_SEMANTIC_ERROR
               (CURRENT_FILE.NEXT-1,
                "Missing digit; inserted ""0"" after ""."""");
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT-1)
               := INSERTED_ZERO_AFTER_DOT;
         else
            SCAN_TO_END_OF_BASED_INTEGER (BASE, DELIMITER);
         end if;
      end if;
   end if;
   if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST then
      if CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) /= DELIMITER then
         SAVE_ERROR_POSITION := CURRENT_FILE.NEXT;
         loop
```

```
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
                  CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               exit when CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = DELIMITER;
               end loop;
               if CURRENT_FILE.NEXT > CURRENT_FILE.LAST then
                  REPORT_SEMANTIC_ERROR
                    (SAVE_ERROR_POSITION,
                     "Expecting based digit or '" & DELIMITER &
                     "'; remainder of line ignored");
                  if SAVE_ERROR_POSITION > CURRENT_FILE.LAST then
                     -- To recover, extend line and put a delimiter in its place.
                     CURRENT_FILE.LAST := CURRENT_FILE.LAST + 1;
                     CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELIMITER;
                  else
                     -- To recover, put delimiter at end of line.
                     CURRENT_FILE.NEXT := CURRENT_FILE.LAST;
                     CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELIMITER;
                  end if;
               else
                  REPORT_SEMANTIC_ERROR
                    (SAVE_ERROR_POSITION,
                     "Expecting based digit or '" & DELIMITER &
                     "'; character(s) through '" & DELIMITER & "' ignored");
               end if;
            end if;
         else
            REPORT_SEMANTIC_ERROR
               (CURRENT_FILE.NEXT,
                "Expecting based digit or '" & DELIMITER &
                "'; remainder of line ignored");
             -- To recover, extend line and put a delimiter in its place.
             CURRENT_FILE.LAST := CURRENT_FILE.LAST + 1;
             CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELIMITER;
         end if;
         -- Skip delimiter.
         if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST then
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         end if;
         -- exponent is scanned below.
      else
         -- Skip optional part.
         if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '.' then
            -- Make sure this is not just the delimiter double dot.
            if CURRENT_FILE.NEXT + 1 <= CURRENT_FILE.LAST and then
               CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT+1) = '.' then
               -- This is a double dot.  We are done.
               null;
```

```
            else
               SAVE_ERROR_POSITION := CURRENT_FILE.NEXT;
               CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               while CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' loop
                  REPORT_SEMANTIC_ERROR
                     (CURRENT_FILE.NEXT,
                      "Deleted illegal leading underline");
                  CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
                  CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               end loop;
               if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST then
                  if CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) not in '0'..'9' then
                     REPORT_SEMANTIC_ERROR
                        (SAVE_ERROR_POSITION,
                         "Missing digit; inserted ""0"" after "".""");
                     CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_A
                  else
                     SCAN_TO_END_OF_DECIMAL_INTEGER;
                  end if;
               else
                  REPORT_SEMANTIC_ERROR
                     (SAVE_ERROR_POSITION,
                      "Missing digit; inserted ""0"" after "".""");
                  CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTE
               end if;
            end if;
         end if;
         -- exponent is scanned below.
      end if;
      -- Scan optional exponent part
      if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
         (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = 'E' or
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = 'e') then
         CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         -- Skip optional sign.
         if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
            (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '-' or
               CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '+') then
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         end if;
         SAVE_ERROR_POSITION := CURRENT_FILE.NEXT - 1;
         while CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' loop
            REPORT_SEMANTIC_ERROR
               (CURRENT_FILE.NEXT,
                "Deleted illegal leading underline");
            CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
```

```
         end loop;
         if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST then
            if CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) not in '0'..'9' then
               REPORT_SEMANTIC_ERROR
                  (SAVE_ERROR_POSITION,
                   "Missing digit; inserted ""0"" after """ &
                    CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) & """");
               case CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) is
                  when 'E' | 'e' =>
                     CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTE
                  when '+' =>
                     CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTE
                  when '-' =>
                     CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTE
                  when others =>
                     REPORT_SYSTEM_ERROR
                        (SAVE_ERROR_POSITION,
                         "Unexpected character.");
               end case;
            else
               SCAN_TO_END_OF_DECIMAL_INTEGER;
            end if;
         else
            REPORT_SEMANTIC_ERROR
               (SAVE_ERROR_POSITION,
                "Missing digit; inserted ""0"" after """ &
                 CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) & """");
            case CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) is
               when 'E' | 'e' =>
                  CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTER_E;
               when '+' =>
                  CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTER_PI
               when '-' =>
                  CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := INSERTED_ZERO_AFTER_MI
               when others =>
                  REPORT_SYSTEM_ERROR
                     (SAVE_ERROR_POSITION,
                      "Unexpected character.");
            end case;
         end if;
        -- ??? Should check that exponent is non-negative for integer literals.
     end if;
     -- Now, START_OF_NUMERIC points to first character in numeric literal.
     -- CURRENT_FILE.NEXT points to character following the last character
     -- of the numeric literal.  The literal may have the characters
     -- DELETED_CHAR and INSERTED_ZERO_AFTER_DOT, INSERTED_ZERO_AFTER_E, and
     -- INSERTED_ZERO_AFTER_MINUS in the literal which should be processed
     -- here.
     TOKEN := new LEXICAL_TOKEN_RECORD (NUMERIC_LITERAL);
```

```
TOKEN.LINE := CURRENT_FILE.LINE;
TOKEN.START := START_OF_NUMERIC;
-- Get length of literal.
STRING_LENGTH := 0;
for I in START_OF_NUMERIC .. CURRENT_FILE.NEXT-1 loop
   case CURRENT_FILE.BUFFER(I) is
      when DELETED_CHAR => null;
      when INSERTED_ZERO_AFTER_DOT | INSERTED_ZERO_AFTER_E |
           INSERTED_ZERO_AFTER_MINUS | INSERTED_ZERO_AFTER_PLUS |
           INSERTED_ZERO_AFTER_ONE | INSERTED_ZERO_AFTER_SHARP |
           INSERTED_ZERO_AFTER_COLON =>
         STRING_LENGTH := STRING_LENGTH + 2;
      when others =>
         STRING_LENGTH := STRING_LENGTH + 1;
   end case;
end loop;
TOKEN.IMAGE := new STRING (1..STRING_LENGTH);
INDEX := 1;
for I in START_OF_NUMERIC .. CURRENT_FILE.NEXT-1 loop
   case CURRENT_FILE.BUFFER(I) is
      when DELETED_CHAR => null;
      when INSERTED_ZERO_AFTER_SHARP =>
         TOKEN.IMAGE.all(INDEX) := '#';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_COLON =>
         TOKEN.IMAGE.all(INDEX) := ':';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_DOT =>
         TOKEN.IMAGE.all(INDEX) := '.';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_ONE =>
         TOKEN.IMAGE.all(INDEX) := '1';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_E =>
         TOKEN.IMAGE.all(INDEX) := 'E';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_MINUS =>
         TOKEN.IMAGE.all(INDEX) := '-';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
      when INSERTED_ZERO_AFTER_PLUS =>
         TOKEN.IMAGE.all(INDEX) := '+';
         TOKEN.IMAGE.all(INDEX+1) := '0';
         INDEX := INDEX + 2;
```

```
            when others =>
               TOKEN.IMAGE.all(INDEX) := CURRENT_FILE.BUFFER(I);
               INDEX := INDEX + 1;
         end case;
      end loop;
      return TOKEN;
   end GET_NUMERIC_LITERAL;

   function GET_IDENTIFIER_OR_RESERVED_WORD return LEXICAL_TOKEN is
      TOKEN                : LEXICAL_TOKEN;
      START_OF_IDENT       : SOURCE_POSITION := CURRENT_FILE.NEXT;
      KIND                 : RESERVED_WORD_KIND;
      FOUND                : BOOLEAN;
      SAVE_ERROR_POSITION  : SOURCE_POSITION;
      STRING_LENGTH        : NATURAL;
      INDEX                : NATURAL;

      procedure UPPER_CASE
         (S : in out STRING) is
      begin
         for I in S'RANGE loop
            if S(I) in 'a'..'z' then
               S(I) := CHARACTER'VAL (CHARACTER'POS (S(I)) - 32);
            end if;
         end loop;
      end UPPER_CASE;

      function IS_ALPHABETIC
         (C : in CHARACTER) return BOOLEAN is
      begin
         if C in 'a'..'z' or C in 'A'..'Z' then
            return TRUE;
         else
            return FALSE;
         end if;
      end IS_ALPHABETIC;

      function IS_ALPHANUMERIC
         (C : in CHARACTER) return BOOLEAN is
      begin
         if C in 'a'..'z' or C in 'A'..'Z' or C in '0'..'9' then
            return TRUE;
         else
            return FALSE;
         end if;
      end IS_ALPHANUMERIC;

      procedure SEARCH_FOR_RESERVED_WORD
         (IDENTIFIER : in  STRING;
```

```
        FOUND       : out BOOLEAN;
        KIND        : out RESERVED_WORD_KIND) is
    begin
       -- The following assumes that RESERVED_WORD_KIND is of the form
       -- "R_" & reserved_word.

       KIND := RESERVED_WORD_KIND'VALUE("R_" & IDENTIFIER);

       -- The above should raise an exception if no such enumeration
       -- literal exists.
       FOUND := TRUE;

    exception
       when OTHERS =>
          FOUND := FALSE;
    end SEARCH_FOR_RESERVED_WORD;

begin
    -- First, scan the identifier and then determine whether it is a
    -- reserved word.
    --
    -- identifier ::= letter {[underline] letter_or_digit}
    -- letter_or_digit := letter | digit
    -- letter ::= upper_case_letter | lower_case_letter

    if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
       IS_ALPHABETIC (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) then
       CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
    else
       REPORT_SYSTEM_ERROR
          (CURRENT_FILE.NEXT,
           "Expecting beginning of identifier.");
    end if;
    loop
       -- Skip optional underline.
       if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
          CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' then
          SAVE_ERROR_POSITION := CURRENT_FILE.NEXT;
          CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
          -- A letter_or_digit must follow an underline.  Verify
          while CURRENT_FILE.NEXT <= CURRENT_FILE.LAST and then
             CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) = '_' loop
             REPORT_SEMANTIC_ERROR
                (CURRENT_FILE.NEXT,
                 "Illegal double underline; deleted ""_""");
             CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT) := DELETED_CHAR;
             CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
          end loop;
          if CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
```

```
            not IS_ALPHANUMERIC (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) then
            REPORT_SEMANTIC_ERROR
               (SAVE_ERROR_POSITION,
                "Deleted illegal trailing underline");
            CURRENT_FILE.BUFFER(SAVE_ERROR_POSITION) := DELETED_CHAR;
         end if;
      end if;
   exit when CURRENT_FILE.NEXT > CURRENT_FILE.LAST or else
      not IS_ALPHANUMERIC (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT));

      CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
   end loop;
   -- Now, START_OF_IDENT points to first character in identifer.
   -- CURRENT_FILE.NEXT points to character following the last character
   -- of the identifier.
   --
   SEARCH_FOR_RESERVED_WORD
      (CURRENT_FILE.BUFFER(START_OF_IDENT..CURRENT_FILE.NEXT-1),
       FOUND,
       KIND);

   if FOUND then
      -- This is a reserved word of kind KIND.
      TOKEN := new LEXICAL_TOKEN_RECORD (RESERVED_WORD);
      TOKEN.LINE := CURRENT_FILE.LINE;
      TOKEN.START := START_OF_IDENT;
      TOKEN.RESERVED_WORD := KIND;
   else
      -- This is a simple identifier.
      TOKEN := new LEXICAL_TOKEN_RECORD (IDENTIFIER);
      TOKEN.LINE := CURRENT_FILE.LINE;
      TOKEN.START := START_OF_IDENT;
      -- Get length of literal.
      STRING_LENGTH := 0;
      for I in START_OF_IDENT .. CURRENT_FILE.NEXT-1 loop
         if CURRENT_FILE.BUFFER(I) /= DELETED_CHAR then
            STRING_LENGTH := STRING_LENGTH + 1;
         end if;
      end loop;
      TOKEN.ID := new STRING (1..STRING_LENGTH);
      INDEX := 1;
      for I in START_OF_IDENT .. CURRENT_FILE.NEXT-1 loop
         if CURRENT_FILE.BUFFER(I) /= DELETED_CHAR then
            TOKEN.ID.all(INDEX) := CURRENT_FILE.BUFFER(I);
            INDEX := INDEX + 1;
         end if;
      end loop;
      UPPER_CASE (TOKEN.ID.all);
   end if;
```

```
      return TOKEN;
   end GET_IDENTIFIER_OR_RESERVED_WORD;

   function GET_DELIMITER return LEXICAL_TOKEN is
      TOKEN : LEXICAL_TOKEN;
      C     : CHARACTER := CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT);
   begin
      TOKEN := new LEXICAL_TOKEN_RECORD (DELIMITER);
      TOKEN.LINE := CURRENT_FILE.LINE;
      TOKEN.START := CURRENT_FILE.NEXT;
      case C is
         when '&' => TOKEN.DELIMITER := AMPERSAND;
         when ''' => TOKEN.DELIMITER := APOSTROPHE;
         when '(' => TOKEN.DELIMITER := LEFT_PARENTHESIS;
         when ')' => TOKEN.DELIMITER := RIGHT_PARENTHESIS;
         when '*' => TOKEN.DELIMITER := STAR;  -- may change.
         when '+' => TOKEN.DELIMITER := PLUS;
         when ',' => TOKEN.DELIMITER := COMMA;
         when '-' => TOKEN.DELIMITER := HYPHEN;
         when '.' => TOKEN.DELIMITER := DOT;  -- may change.
         when '/' => TOKEN.DELIMITER := SLASH;  -- may change.
         when ':' => TOKEN.DELIMITER := COLON;  -- may change.
         when ';' => TOKEN.DELIMITER := SEMICOLON;
         when '<' => TOKEN.DELIMITER := LESS_THAN;  -- may change.
         when '=' => TOKEN.DELIMITER := EQUAL;      -- may change.
         when '>' => TOKEN.DELIMITER := GREATER_THAN;  -- may change.
         when '|' => TOKEN.DELIMITER := VERTICAL_BAR;
         when '!' => TOKEN.DELIMITER := VERTICAL_BAR;
         when others =>
            REPORT_SYSTEM_ERROR (CURRENT_FILE.NEXT, "Expecting delimiter.");
      end case;
      CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
      if CURRENT_FILE.NEXT <= CURRENT_FILE.LAST then
         -- Check to see if compound delimiter.
         C := CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT);
         case TOKEN.DELIMITER is
            when EQUAL =>
               if C = '>' then
                  TOKEN.DELIMITER := ARROW;
                  CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               end if;
            when DOT =>
               if C = '.' then
                  TOKEN.DELIMITER := DOUBLE_DOT;
                  CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
               end if;
            when STAR =>
               if C = '*' then
                  TOKEN.DELIMITER := DOUBLE_STAR;
```

```
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             end if;
          when COLON =>
             if C = '=' then
                TOKEN.DELIMITER := ASSIGNMENT;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             end if;
          when SLASH =>
             if C = '=' then
                TOKEN.DELIMITER := INEQUALITY;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             end if;
          when GREATER_THAN =>
             if C = '=' then
                TOKEN.DELIMITER := GREATER_THAN_OR_EQUAL;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             elsif C = '>' then
                TOKEN.DELIMITER := RIGHT_LABEL_BRACKET;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             end if;
          when LESS_THAN =>
             if C = '=' then
                TOKEN.DELIMITER := LESS_THAN_OR_EQUAL;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             elsif C = '<' then
                TOKEN.DELIMITER := LEFT_LABEL_BRACKET;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             elsif C = '>' then
                TOKEN.DELIMITER := BOX;
                CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
             end if;
          when others =>
             null;
       end case;
    end if;
    return TOKEN;
 end GET_DELIMITER;

 function GET_NEXT_TOKEN return LEXICAL_TOKEN is
    TOKEN : LEXICAL_TOKEN;

    function PRINTABLE_IMAGE
       (C : in CHARACTER) return STRING is
    begin
       case C is
          when ASCII.NUL => return "ASCII.NUL";
          when ASCII.STX => return "ASCII.STX";
          when ASCII.EOT => return "ASCII.EOT";
          when ASCII.ACK => return "ASCII.ACK";
```

```
            when ASCII.BS  => return "ASCII.BS";
            when ASCII.LF  => return "ASCII.LF";
            when ASCII.FF  => return "ASCII.FF";
            when ASCII.SO  => return "ASCII.SO";
            when ASCII.DLE => return "ASCII.DLE";
            when ASCII.DC2 => return "ASCII.DC2";
            when ASCII.DC4 => return "ASCII.DC4";
            when ASCII.SYN => return "ASCII.SYN";
            when ASCII.CAN => return "ASCII.CAN";
            when ASCII.SUB => return "ASCII.SUB";
            when ASCII.FS  => return "ASCII.FS";
            when ASCII.RS  => return "ASCII.RS";
            when ASCII.DEL => return "ASCII.DEL";
            when ASCII.SOH => return "ASCII.SOH";
            when ASCII.ETX => return "ASCII.ETX";
            when ASCII.ENQ => return "ASCII.ENQ";
            when ASCII.BEL => return "ASCII.BEL";
            when ASCII.HT  => return "ASCII.HT";
            when ASCII.VT  => return "ASCII.VT";
            when ASCII.CR  => return "ASCII.CR";
            when ASCII.SI  => return "ASCII.SI";
            when ASCII.DC1 => return "ASCII.DC1";
            when ASCII.DC3 => return "ASCII.DC3";
            when ASCII.NAK => return "ASCII.NAK";
            when ASCII.ETB => return "ASCII.ETB";
            when ASCII.EM  => return "ASCII.EM";
            when ASCII.ESC => return "ASCII.ESC";
            when ASCII.GS  => return "ASCII.GS";
            when ASCII.US  => return "ASCII.US";
            when others    => return "'" & C & "'";
         end case;
      end PRINTABLE_IMAGE;

   begin
      loop  -- until we have a token.
         ADVANCE_TO_NEXT_TOKEN;
         if CURRENT_FILE.EOF then
            TOKEN := GET_EOF_TOKEN;
         elsif START_OF_CHARACTER_LITERAL then
            TOKEN := GET_CHARACTER_LITERAL;
         elsif START_OF_STRING_LITERAL then
            TOKEN := GET_STRING_LITERAL;
         elsif START_OF_NUMERIC_LITERAL then
            TOKEN := GET_NUMERIC_LITERAL;
         elsif START_OF_IDENTIFIER then
            TOKEN := GET_IDENTIFIER_OR_RESERVED_WORD;
         elsif START_OF_DELIMITER then
            TOKEN := GET_DELIMITER;
         else
```

```
            -- Skip over unknown lexical element.
            REPORT_SEMANTIC_ERROR
               (CURRENT_FILE.NEXT,
                "Illegal character (" &
                PRINTABLE_IMAGE (CURRENT_FILE.BUFFER(CURRENT_FILE.NEXT)) &
                ") ignored");
            CURRENT_FILE.NEXT := CURRENT_FILE.NEXT + 1;
         end if;
      exit when TOKEN /= null;
      end loop;
      if IGNORE_PRAGMAS then
         -- must check if this token is a pragma.
         if TOKEN.KIND = RESERVED_WORD and then
            TOKEN.RESERVED_WORD = R_PRAGMA then
            -- we have a pragma token.
            -- pragma ::= pragma identifier [(argument_association {,
            --                  argument_association})];
            -- argument_association ::= [argument_identifier =>] name
            --      | [argument_identifier =>] expression
            --
            -- for simplicity here, we just skip to the semicolon.
            loop
               TOKEN := GET_NEXT_TOKEN;
               if TOKEN.KIND = END_OF_FILE then
                  REPORT_FATAL_ERROR
                     (CURRENT_FILE.NEXT,
                      "Premature end of file encountered.");
               end if;
            exit when TOKEN.KIND = DELIMITER and then
               TOKEN.DELIMITER = SEMICOLON;
            end loop;
            TOKEN := GET_NEXT_TOKEN;  -- skip semicolon.
         end if;
      end if;
      return TOKEN;
   end GET_NEXT_TOKEN;

   function NEXT_TOKEN return LEXICAL_TOKEN is
      TOKEN_ENTRY : TOKEN_LIST_ENTRY;
      TOKEN       : LEXICAL_TOKEN;
   begin
      if CURRENT_FILE = null or else not CURRENT_FILE.IS_OPEN then
         REPORT_FATAL_ERROR ("Internal system error -- File not open.");
      end if;
      if CURRENT_FILE.LOOK_AHEAD_TOKENS /= null then
         -- Take token from front of list.
         TOKEN_ENTRY := CURRENT_FILE.LOOK_AHEAD_TOKENS;
         CURRENT_FILE.LOOK_AHEAD_TOKENS := CURRENT_FILE.LOOK_AHEAD_TOKENS.NEXT;
         TOKEN := TOKEN_ENTRY.TOKEN;
```

```
         FREE (TOKEN_ENTRY);
      else
         -- No look-ahead tokens.  Get the next token from the file.
         TOKEN := GET_NEXT_TOKEN;
      end if;
      -- Reset the look-ahead pointer.
      CURRENT_FILE.LOOK_AHEAD_PTR := 0;
      -- PRINT_TOKEN (TOKEN);
      return TOKEN;
   end NEXT_TOKEN;

   function NEXT_LOOK_AHEAD_TOKEN return LEXICAL_TOKEN is
      COUNT   : INTEGER;
      TRACER : TOKEN_LIST_ENTRY;
   begin
      if CURRENT_FILE = null or else not CURRENT_FILE.IS_OPEN then
         REPORT_FATAL_ERROR ("Internal system error -- File not open.");
      end if;

      CURRENT_FILE.LOOK_AHEAD_PTR := CURRENT_FILE.LOOK_AHEAD_PTR + 1;

      -- Now count number of look-ahead tokens that are stored away already.
      COUNT := 0;
      TRACER := CURRENT_FILE.LOOK_AHEAD_TOKENS;
      while TRACER /= null loop
         COUNT := COUNT + 1;
         TRACER := TRACER.NEXT;
      end loop;

      if CURRENT_FILE.LOOK_AHEAD_PTR <= COUNT then
         -- We already have read the token requested.
         TRACER := CURRENT_FILE.LOOK_AHEAD_TOKENS;
         for I in 1..CURRENT_FILE.LOOK_AHEAD_PTR-1 loop
            TRACER := TRACER.NEXT;
         end loop;
         -- PRINT_TOKEN (TRACER.TOKEN);
         return TRACER.TOKEN;
      elsif CURRENT_FILE.LOOK_AHEAD_PTR = COUNT + 1 then
         -- We need to add one more look-ahead token to our list.
         -- Find end of list.
         if CURRENT_FILE.LOOK_AHEAD_TOKENS = null then
            -- List is empty.
            -- Need to get a look-ahead token.
            CURRENT_FILE.LOOK_AHEAD_TOKENS := new TOKEN_LIST_ENTRY_RECORD;
            CURRENT_FILE.LOOK_AHEAD_TOKENS.TOKEN := GET_NEXT_TOKEN;
            -- PRINT_TOKEN (CURRENT_FILE.LOOK_AHEAD_TOKENS.TOKEN);
            return CURRENT_FILE.LOOK_AHEAD_TOKENS.TOKEN;
         else
            TRACER := CURRENT_FILE.LOOK_AHEAD_TOKENS;
```

```
            while TRACER.NEXT /= null loop
               TRACER := TRACER.NEXT;
            end loop;
            TRACER.NEXT := new TOKEN_LIST_ENTRY_RECORD;
            TRACER.NEXT.TOKEN := GET_NEXT_TOKEN;
            -- PRINT_TOKEN (TRACER.NEXT.TOKEN);
            return TRACER.NEXT.TOKEN;
         end if;
      else
         -- Something is wrong with our count.
         REPORT_SYSTEM_ERROR
            (CURRENT_FILE.NEXT,
             "Something is wrong with count in NEXT_LOOK_AHEAD_TOKEN.");
      end if;
   end NEXT_LOOK_AHEAD_TOKEN;

   procedure SET_LOOK_AHEAD is
   begin
      if CURRENT_FILE = null or else not CURRENT_FILE.IS_OPEN then
         REPORT_FATAL_ERROR ("Internal system error -- File not open.");
      end if;
      CURRENT_FILE.LOOK_AHEAD_PTR := 0;
   end SET_LOOK_AHEAD;

   function FIRST_LOOK_AHEAD_TOKEN return LEXICAL_TOKEN is
      TOKEN : LEXICAL_TOKEN;
   begin
      SET_LOOK_AHEAD;
      TOKEN := NEXT_LOOK_AHEAD_TOKEN;
      -- PRINT_TOKEN (TOKEN);
      return TOKEN;
   end FIRST_LOOK_AHEAD_TOKEN;

   procedure EAT_NEXT_TOKEN is
      TOKEN : LEXICAL_TOKEN;
   begin
      TOKEN := NEXT_TOKEN;
   end EAT_NEXT_TOKEN;

   procedure SKIP_TOKEN_FOR_NOW is
      TOKEN_ENTRY : TOKEN_LIST_ENTRY;
   begin
      TOKEN_ENTRY := new TOKEN_LIST_ENTRY_RECORD;
      TOKEN_ENTRY.TOKEN := NEXT_TOKEN;
      TOKEN_ENTRY.NEXT := CURRENT_FILE.SKIPPED_TOKENS;
      CURRENT_FILE.SKIPPED_TOKENS := TOKEN_ENTRY;
   end SKIP_TOKEN_FOR_NOW;

   procedure RESTORE_SKIPPED_TOKENS is
```

```
   TOKEN_ENTRY : TOKEN_LIST_ENTRY;
begin
   if CURRENT_FILE = null or else not CURRENT_FILE.IS_OPEN then
      REPORT_FATAL_ERROR ("Internal system error -- File not open.");
   end if;
   while CURRENT_FILE.SKIPPED_TOKENS /= null loop
      -- Remove most-recently skipped token (first).
      TOKEN_ENTRY := CURRENT_FILE.SKIPPED_TOKENS;
      CURRENT_FILE.SKIPPED_TOKENS := CURRENT_FILE.SKIPPED_TOKENS.NEXT;
      -- Put skipped token on look-ahead list.
      TOKEN_ENTRY.NEXT := CURRENT_FILE.LOOK_AHEAD_TOKENS;
      CURRENT_FILE.LOOK_AHEAD_TOKENS := TOKEN_ENTRY;
   end loop;
end RESTORE_SKIPPED_TOKENS;

procedure ZAP_SKIPPED_TOKENS is
   CURRENT, NEXT : TOKEN_LIST_ENTRY;
begin
   if CURRENT_FILE = null or else not CURRENT_FILE.IS_OPEN then
      REPORT_FATAL_ERROR ("Internal system error -- File not open.");
   end if;
   CURRENT := CURRENT_FILE.SKIPPED_TOKENS;
   while CURRENT /= null loop
      NEXT := CURRENT.NEXT;
      FREE ( CURRENT );
      CURRENT := NEXT;
   end loop;
   CURRENT_FILE.SKIPPED_TOKENS := null;
end ZAP_SKIPPED_TOKENS;

procedure PRINT_HEADING is
begin
   if CURRENT_FILE.USE_STANDARD_OUTPUT then
      TEXT_IO.NEW_PAGE;
      TEXT_IO.PUT ("Ada/SQL Application Scanner Listing");
      TEXT_IO.SET_COL (70);
      TEXT_IO.PUT_LINE ("Page " &
                  INTEGER'IMAGE(CURRENT_FILE.ERROR_FILE_PAGE));
      TEXT_IO.NEW_LINE;
   else
      TEXT_IO.NEW_PAGE (CURRENT_FILE.ERROR_FILE);
      TEXT_IO.PUT (CURRENT_FILE.ERROR_FILE,
             "Ada/SQL Application Scanner Listing");
      TEXT_IO.SET_COL (CURRENT_FILE.ERROR_FILE, 70);
      TEXT_IO.PUT_LINE (CURRENT_FILE.ERROR_FILE, "Page " &
                  INTEGER'IMAGE(CURRENT_FILE.ERROR_FILE_PAGE));
      TEXT_IO.NEW_LINE (CURRENT_FILE.ERROR_FILE);
   end if;
   CURRENT_FILE.ERROR_FILE_LINE := 2;
```

```
end PRINT_HEADING;

procedure SET_INDENT
   (COUNT       : TEXT_IO.POSITIVE_COUNT;
    TO_TERMINAL : BOOLEAN := FALSE) is
begin
   if TO_TERMINAL or else CURRENT_FILE.USE_STANDARD_OUTPUT then
      TEXT_IO.SET_COL (COUNT);
   else
      TEXT_IO.SET_COL (CURRENT_FILE.ERROR_FILE,  COUNT);
   end if;
end SET_INDENT;

procedure PRINT
   (LINE        : in STRING;
    TO_TERMINAL : in BOOLEAN := FALSE) is
begin
   if not TO_TERMINAL and then CURRENT_FILE.ERROR_FILE_LINE = 0 then
      PRINT_HEADING;
   end if;
   if TO_TERMINAL or else CURRENT_FILE.USE_STANDARD_OUTPUT then
      TEXT_IO.PUT_LINE (LINE);
   else
      TEXT_IO.PUT_LINE (CURRENT_FILE.ERROR_FILE, LINE);
   end if;
   if not TO_TERMINAL then
      CURRENT_FILE.ERROR_FILE_LINE := CURRENT_FILE.ERROR_FILE_LINE + 1;
      if CURRENT_FILE.ERROR_FILE_LINE > LINES_PER_PAGE_FOR_ERROR_LISTING then
         CURRENT_FILE.ERROR_FILE_PAGE := CURRENT_FILE.ERROR_FILE_PAGE + 1;
         CURRENT_FILE.ERROR_FILE_LINE := 0;
      end if;
   end if;
end PRINT;

procedure BREAK_LINE
   (LINE  : in STRING;
    FIRST : in NATURAL;
    LAST  : in out NATURAL;
    NEW_FIRST : out NATURAL) is

   ORIGINAL_LAST : NATURAL := LAST;
   LOCAL_NEW_FIRST : NATURAL;
begin
   if LINE'LAST > LAST then
      LAST := LAST + 1;
   end if;
   while LAST > FIRST and then LINE(LAST) /= ' ' loop
      LAST := LAST - 1;
   end loop;
```

```
   if LAST = FIRST then
      -- line has no natural breaking point.  Just break at end.
      LAST := ORIGINAL_LAST;
   else
      -- LAST points at blank.  Adjust back one.
      LAST := LAST - 1;
   end if;
   -- Find new first.
   LOCAL_NEW_FIRST := LAST + 1;
   while LOCAL_NEW_FIRST <= LINE'LAST and then
      LINE(LOCAL_NEW_FIRST) = ' ' loop
      LOCAL_NEW_FIRST := LOCAL_NEW_FIRST + 1;
   end loop;
   NEW_FIRST := LOCAL_NEW_FIRST;
end BREAK_LINE;

procedure DISPLAY_LINE
   (LINE        : in STRING;
    WRAP        : in BOOLEAN := FALSE;
    TO_TERMINAL : in BOOLEAN := FALSE) is
    -- if WRAP = TRUE, then wrap lines at 80 columns with an indent of 5
    -- characters for each subsequent line.
    -- if WRAP = FALSE, then truncate lines at 132 columns.
   LAST      : NATURAL;
   FIRST     : NATURAL;
   NEW_FIRST : NATURAL;
begin
   if not WRAP or else LINE'LENGTH <= MESSAGE_WRAP_LENGTH then
      -- truncate line at 132.
      if LINE'LENGTH > COLUMNS_PER_LINE_FOR_ERROR_LISTING then
         LAST := LINE'FIRST + COLUMNS_PER_LINE_FOR_ERROR_LISTING - 1;
      else
         LAST := LINE'LAST;
      end if;
      PRINT (LINE(LINE'FIRST..LAST), TO_TERMINAL => TO_TERMINAL);
   else
      -- Line is at least MESSAGE_WRAP_LENGTH.
      FIRST := LINE'FIRST;
      LAST := LINE'LAST;
      BREAK_LINE (LINE, FIRST, LAST, NEW_FIRST);
      PRINT (LINE(FIRST..LAST), TO_TERMINAL => TO_TERMINAL);
      while NEW_FIRST <= LINE'LAST loop
         -- Need to continue line.
         if NEW_FIRST - LINE'LAST + 1 + INDENT_FOR_MESSAGE_WRAP
            > MESSAGE_WRAP_LENGTH then
            -- Need to break line.
            LAST := NEW_FIRST + MESSAGE_WRAP_LENGTH - 1;
            FIRST := NEW_FIRST;
            BREAK_LINE (LINE, FIRST, LAST, NEW_FIRST);
```

```
                SET_INDENT (TEXT_IO.COUNT(INDENT_FOR_MESSAGE_WRAP),
                            TO_TERMINAL => TO_TERMINAL);
                PRINT (LINE (FIRST..LAST), TO_TERMINAL => TO_TERMINAL);
             else
                SET_INDENT (TEXT_IO.COUNT(INDENT_FOR_MESSAGE_WRAP),
                            TO_TERMINAL => TO_TERMINAL);
                PRINT (LINE (NEW_FIRST..LINE'LAST), TO_TERMINAL => TO_TERMINAL);
                NEW_FIRST := LINE'LAST + 1;
             end if;
          end loop;
       end if;
    end DISPLAY_LINE;

    procedure DISPLAY_ERROR
       (KIND : in MESSAGE_KIND;
        LINE : in SOURCE_LINE;
        COL  : in SOURCE_POSITION;
        MESS : in STRING;
        TO_TERMINAL : in BOOLEAN := FALSE) is
       LINE_BUFF : STRING (1..MAXIMUM_INPUT_LINE_LENGTH+6) := (others => ' ');
       K     : CHARACTER;
    begin
--          if not TO_TERMINAL and then KIND /= FATAL then
       if not TO_TERMINAL and then KIND /= FATAL and then
               not (LINE = 0 or COL = MAXIMUM_INPUT_LINE_LENGTH) then
          LINE_BUFF(COL+6) := '^';
          DISPLAY_LINE (LINE_BUFF);
       end if;
       case KIND is
          when SYNTAX | SEMANTIC => K := 'E';
          when SYSTEM            => K := 'S';
          when FATAL             => K := 'F';
          when WARNING           => K := 'W';
          when NOTE              => K := 'I';
       end case;
       if LINE = 0 and then COL = MAXIMUM_INPUT_LINE_LENGTH then
          DISPLAY_LINE (MESS, WRAP => TRUE, TO_TERMINAL => TO_TERMINAL);
       elsif TO_TERMINAL and then LINE /= 0 then
          DISPLAY_LINE ("%ADASQL-" & K & "-SCAN" &
             " on line " & SOURCE_LINE'IMAGE(LINE) & ", " & MESS, WRAP => TRUE,
             TO_TERMINAL => TO_TERMINAL);
       else
          DISPLAY_LINE ("%ADASQL-" & K & "-SCAN" & ", " & MESS, WRAP => TRUE,
             TO_TERMINAL => TO_TERMINAL);
       end if;
    end DISPLAY_ERROR;

    procedure REPORT_ERROR
       (KIND : in MESSAGE_KIND;
```

```
     LINE : in SOURCE_LINE;
     COL  : in SOURCE_POSITION;
     MESS : in STRING) is

   MESSAGE_ENTRY   : MESSAGE_LIST_ENTRY;
   LOCAL_COL       : SOURCE_POSITION := COL;

   procedure ADD_TO_MESSAGE_LIST
      (MESS_ENTRY : in MESSAGE_LIST_ENTRY) is
      -- Insert into list in ascending order of line and column.
      TRACER : MESSAGE_LIST_ENTRY := CURRENT_FILE.MESSAGE_LIST;
   begin
      -- Find insertion point.
      if TRACER = null then
         -- List is empty.  Add.
         MESS_ENTRY.NEXT := null;
         CURRENT_FILE.MESSAGE_LIST := MESS_ENTRY;
      elsif TRACER.LINE > MESS_ENTRY.LINE or else
         (TRACER.LINE = MESS_ENTRY.LINE and TRACER.START > MESS_ENTRY.START)
         -- Insert in front of list.
         MESS_ENTRY.NEXT := CURRENT_FILE.MESSAGE_LIST;
         CURRENT_FILE.MESSAGE_LIST := MESS_ENTRY;
      else
         while TRACER.NEXT /= null and then
            (TRACER.NEXT.LINE < MESS_ENTRY.LINE or else
            (TRACER.NEXT.LINE = MESS_ENTRY.LINE and TRACER.NEXT.START <= MESS_
            TRACER := TRACER.NEXT;
         end loop;
         MESS_ENTRY.NEXT := TRACER.NEXT;
         TRACER.NEXT := MESS_ENTRY;
      end if;
   end ADD_TO_MESSAGE_LIST;

begin
   if COL <= 0 then
      LOCAL_COL := 1;
   end if;
   if DISPLAY_ERRORS_IMMEDIATELY or else CURRENT_FILE = null or else
      KIND = FATAL then
      DISPLAY_ERROR (KIND, LINE, LOCAL_COL, MESS, TO_TERMINAL => TRUE);
   end if;

   if CURRENT_FILE = null then
      return;
   end if;

   if CURRENT_FILE.ERROR_FILENAME.all /= "" then
      MESSAGE_ENTRY := new MESSAGE_LIST_ENTRY_RECORD;
      MESSAGE_ENTRY.LINE := LINE;
```

```
      MESSAGE_ENTRY.START := LOCAL_COL;
      MESSAGE_ENTRY.KIND := KIND;
      MESSAGE_ENTRY.MESSAGE := new STRING(1..MESS'LENGTH);
      MESSAGE_ENTRY.MESSAGE.all := MESS;
      -- Add entry to messages for current file.
      ADD_TO_MESSAGE_LIST (MESSAGE_ENTRY);
   end if;

   CURRENT_FILE.ERROR_COUNT(KIND) := CURRENT_FILE.ERROR_COUNT(KIND) + 1;

   if KIND /= FATAL and then
      SEVERE_ERRORS > MAXIMUM_NUMBER_OF_ERRORS then
      REPORT_FATAL_ERROR
         (CURRENT_FILE.NEXT,
          "Terminating scan since ERROR_LIMIT=" &
           INTEGER'IMAGE(MAXIMUM_NUMBER_OF_ERRORS) & " reached");
   end if;

end REPORT_ERROR;

procedure REPORT_DDL_ERROR
   (MESSAGE : in STRING) is
begin
  REPORT_ERROR (SYNTAX, 0, MAXIMUM_INPUT_LINE_LENGTH, MESSAGE);
end REPORT_DDL_ERROR;

procedure REPORT_SYNTAX_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (SYNTAX, CURRENT_FILE.LINE, COL, MESSAGE);
   ZAP_SKIPPED_TOKENS;
   raise SYNTAX_ERROR;
end REPORT_SYNTAX_ERROR;

procedure REPORT_SYNTAX_ERROR
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (SYNTAX, TOKEN.LINE, TOKEN.START, MESSAGE);
   ZAP_SKIPPED_TOKENS;
   raise SYNTAX_ERROR;
end REPORT_SYNTAX_ERROR;

procedure REPORT_SEMANTIC_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (SEMANTIC, CURRENT_FILE.LINE, COL, MESSAGE);
end REPORT_SEMANTIC_ERROR;
```

```
procedure REPORT_SEMANTIC_ERROR
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (SEMANTIC, TOKEN.LINE, TOKEN.START, MESSAGE);
end REPORT_SEMANTIC_ERROR;

procedure REPORT_FATAL_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (FATAL, CURRENT_FILE.LINE, COL, MESSAGE);
   if CURRENT_FILE /= null or CURRENT_FILE.IS_OPEN then
     ZAP_SKIPPED_TOKENS;
   end if;
   raise FATAL_ERROR;
end REPORT_FATAL_ERROR;

procedure REPORT_FATAL_ERROR
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (FATAL, TOKEN.LINE, TOKEN.START, MESSAGE);
   if CURRENT_FILE /= null or CURRENT_FILE.IS_OPEN then
     ZAP_SKIPPED_TOKENS;
   end if;
   raise FATAL_ERROR;
end REPORT_FATAL_ERROR;

procedure REPORT_FATAL_ERROR
   (MESSAGE : in STRING) is
begin
   if CURRENT_FILE = null then
      REPORT_ERROR (FATAL, 0, 0, MESSAGE);
   else
      REPORT_ERROR (FATAL, CURRENT_FILE.LINE, 0, MESSAGE);
   end if;
   if CURRENT_FILE /= null or CURRENT_FILE.IS_OPEN then
     ZAP_SKIPPED_TOKENS;
   end if;
   raise FATAL_ERROR;
end REPORT_FATAL_ERROR;

procedure REPORT_SYSTEM_ERROR
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (SYSTEM, CURRENT_FILE.LINE, COL, MESSAGE);
   ZAP_SKIPPED_TOKENS;
   raise SYSTEM_ERROR;
end REPORT_SYSTEM_ERROR;
```

```
procedure REPORT_SYSTEM_ERROR
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (SYSTEM, TOKEN.LINE, TOKEN.START, MESSAGE);
   ZAP_SKIPPED_TOKENS;
   raise SYSTEM_ERROR;
end REPORT_SYSTEM_ERROR;

procedure REPORT_SYSTEM_ERROR
   (MESSAGE : in STRING) is
begin
   if CURRENT_FILE = null then
      REPORT_ERROR (SYSTEM, 0, 0, MESSAGE);
   else
      REPORT_ERROR (SYSTEM, CURRENT_FILE.LINE, 0, MESSAGE);
   end if;
   ZAP_SKIPPED_TOKENS;
   raise SYSTEM_ERROR;
end REPORT_SYSTEM_ERROR;

procedure REPORT_WARNING
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (WARNING, CURRENT_FILE.LINE, COL, MESSAGE);
end REPORT_WARNING;

procedure REPORT_WARNING
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (WARNING, TOKEN.LINE, TOKEN.START, MESSAGE);
end REPORT_WARNING;

procedure REPORT_NOTE
   (COL : in SOURCE_POSITION; MESSAGE : in STRING) is
begin
   REPORT_ERROR (NOTE, CURRENT_FILE.LINE, COL, MESSAGE);
end REPORT_NOTE;

procedure REPORT_NOTE
   (TOKEN   : in LEXICAL_TOKEN;
    MESSAGE : in STRING) is
begin
   REPORT_ERROR (NOTE, TOKEN.LINE, TOKEN.START, MESSAGE);
end REPORT_NOTE;

function SEVERE_ERRORS return INTEGER is
begin
```

```
      return CURRENT_FILE.ERROR_COUNT(SYNTAX) +
         CURRENT_FILE.ERROR_COUNT(SEMANTIC) +
         CURRENT_FILE.ERROR_COUNT(SYSTEM) +
         CURRENT_FILE.ERROR_COUNT(FATAL);
   end SEVERE_ERRORS;

   procedure PRODUCE_ERROR_LISTING is
      package INT_IO is new TEXT_IO.INTEGER_IO(INTEGER);
      ERROR_COUNT : STRING (1..5);
   begin
      if CURRENT_FILE = null then
         return;
      end if;
      if CURRENT_FILE.ERROR_COUNT(SYNTAX) /= 0 or else
         CURRENT_FILE.ERROR_COUNT(SEMANTIC) /= 0 then
         DISPLAY_ERROR (NOTE, 0, 0, "Scan completed with errors",
                        TO_TERMINAL => TRUE);
      elsif CURRENT_FILE.ERROR_COUNT(WARNING) /= 0 then
         DISPLAY_ERROR (NOTE, 0, 0, "Scan completed with warnings",
                        TO_TERMINAL => TRUE);
      elsif SEVERE_ERRORS = 0 then
         DISPLAY_ERROR (NOTE, 0, 0, "Scan completed with no errors detected",
                        TO_TERMINAL => TRUE);
      end if;
      if CURRENT_FILE.ERROR_FILENAME.all /= "" then
         if not CURRENT_FILE.USE_STANDARD_OUTPUT then
            -- must create error listing file.
           TEXT_IO.CREATE
              (FILE => CURRENT_FILE.ERROR_FILE,
               NAME => CURRENT_FILE.ERROR_FILENAME.all);
         end if;

         -- Message list is already in ascending order.
         declare
            LINE : STRING (1..MAXIMUM_INPUT_LINE_LENGTH);
            LAST : SOURCE_POSITION;
            CURRENT_LINE : SOURCE_LINE := 0;
            LINE_NUMBER : STRING (1..5);

         begin
            TEXT_IO.RESET (CURRENT_FILE.SHADOW_FILE, TEXT_IO.IN_FILE);
            loop
               while CURRENT_FILE.MESSAGE_LIST /= null and then
                  CURRENT_FILE.MESSAGE_LIST.LINE <= CURRENT_LINE loop
                  DISPLAY_ERROR
                     (CURRENT_FILE.MESSAGE_LIST.KIND,
                      CURRENT_FILE.MESSAGE_LIST.LINE,
                      CURRENT_FILE.MESSAGE_LIST.START,
                      CURRENT_FILE.MESSAGE_LIST.MESSAGE.all);
```

```
               CURRENT_FILE.MESSAGE_LIST := CURRENT_FILE.MESSAGE_LIST.NEXT;
            end loop;
            if CURRENT_LINE < 1 then
              DISPLAY_LINE (" ");
            end if;
            TEXT_IO.GET_LINE (CURRENT_FILE.SHADOW_FILE, LINE, LAST);
            CURRENT_LINE := CURRENT_LINE + 1;
            INT_IO.PUT(LINE_NUMBER, INTEGER(CURRENT_LINE));
            DISPLAY_LINE (LINE_NUMBER & " " & LINE (1..LAST));
         end loop;
         TEXT_IO.CLOSE (CURRENT_FILE.SHADOW_FILE);
      exception
         when others => null;
      end;
      -- Display rest of error messages.
      while CURRENT_FILE.MESSAGE_LIST /= null loop
         DISPLAY_ERROR
            (CURRENT_FILE.MESSAGE_LIST.KIND,
               CURRENT_FILE.MESSAGE_LIST.LINE,
               CURRENT_FILE.MESSAGE_LIST.START,
               CURRENT_FILE.MESSAGE_LIST.MESSAGE.all);
         CURRENT_FILE.MESSAGE_LIST := CURRENT_FILE.MESSAGE_LIST.NEXT;
      end loop;
      -- Display summary information.
      DISPLAY_LINE ("");
      DISPLAY_LINE ("SUMMARY:");
      DISPLAY_LINE ("");
      INT_IO.PUT (ERROR_COUNT, CURRENT_FILE.ERROR_COUNT(SYNTAX) +
         CURRENT_FILE.ERROR_COUNT(SEMANTIC));
      DISPLAY_LINE ("    " & ERROR_COUNT & " errors.");
      INT_IO.PUT (ERROR_COUNT, CURRENT_FILE.ERROR_COUNT(WARNING));
      DISPLAY_LINE ("    " & ERROR_COUNT & " warnings.");
      INT_IO.PUT (ERROR_COUNT, CURRENT_FILE.ERROR_COUNT(FATAL));
      DISPLAY_LINE ("    " & ERROR_COUNT & " fatal errors.");
   end if;
   if not CURRENT_FILE.USE_STANDARD_OUTPUT then
      TEXT_IO.CLOSE (CURRENT_FILE.ERROR_FILE);
   end if;
end PRODUCE_ERROR_LISTING;

procedure PRINT_TOKEN
   (TOKEN : in LEXICAL_TOKEN) is
begin
   if TOKEN = null then
      TEXT_IO.PUT_LINE ("*** NULL_TOKEN ????");
   else
      TEXT_IO.PUT (SOURCE_LINE'IMAGE(TOKEN.LINE));
      TEXT_IO.PUT (" ");
      TEXT_IO.PUT (SOURCE_POSITION'IMAGE(TOKEN.START));
```

```
            TEXT_IO.PUT (" ");
            TEXT_IO.PUT (TOKEN_KIND'IMAGE(TOKEN.KIND));
            TEXT_IO.PUT (" -->");
            case TOKEN.KIND is
               when IDENTIFIER        => TEXT_IO.PUT (TOKEN.ID.all);
               when NUMERIC_LITERAL   => TEXT_IO.PUT (TOKEN.IMAGE.all);
               when CHARACTER_LITERAL => TEXT_IO.PUT (TOKEN.CHARACTER_VALUE);
               when STRING_LITERAL    => TEXT_IO.PUT (TOKEN.STRING_IMAGE.all);
               when DELIMITER         => TEXT_IO.PUT (DELIMITER_KIND'IMAGE(TOKEN.DE
               when RESERVED_WORD     => TEXT_IO.PUT (RESERVED_WORD_KIND'IMAGE(TOKE
               when END_OF_FILE       => TEXT_IO.PUT ("End of file");
            end case;
            TEXT_IO.PUT_LINE ("<--");
         end if;
      end PRINT_TOKEN;

end LEXICAL_ANALYZER;
```

3.11.9 package ddl_io_defs_spec.ada

```
with TEXT_IO;
use  TEXT_IO;

package IO_DEFINITIONS is

  type INPUT_RECORD is
    record
      ORIG_BUF : STRING (1..200) := (others => ' ');
      BUFFER   : STRING (1..200) := (others => ' ');
      FILE     : FILE_TYPE;
      START    : POSITIVE := 1;
      NEXT     : POSITIVE := 1;
      LAST     : NATURAL := 0;
      LINE     : NATURAL := 0;
    end record;

  type INPUT_STREAM is access INPUT_RECORD;
  type HOW_TO_DO_FILES_TYPE is (UPPER_CASE, LOWER_CASE, AS_IS);
  type SCHEMA_FROM is (FILES, CALLS, UNKNOWN);

  OUTPUT_FILE_TYPE        : FILE_TYPE;
  FATAL_ERRORS            : NATURAL := 0;
  OUTPUT_FILE_IS_OPEN     : BOOLEAN := FALSE;
  OUTPUT_FILE_NAME        : STRING (1..250) := (others => ' ');
  OUTPUT_FILE_NAME_LEN    : NATURAL := 0;
  OFN_EXTEN               : constant STRING := ".DDLOUT";
  OFN_EXTEN_LEN           : constant INTEGER := 7;
  CALLED_STANDARD_YET     : BOOLEAN := FALSE;

-- standard_name_file is as the file name should be accessed, without extention
```

```
-- standard_name is the package name
-- standard_name_ada_sql is the nexted package name

  function STANDARD_NAME_FILE return STRING;
  --STANDARD_NAME_FILE     : constant STRING := "ADASQL$ENV:STANDARD";
  STANDARD_NAME          : constant STRING := "STANDARD";
  STANDARD_NAME_ADA_SQL : constant STRING := "STANDARD";

-- cursor_name_file is as the file name should be accessed, without extention
-- cursor_name is the package name
-- cursor_name_ada_sql is the nexted package name

  function CURSOR_NAME_FILE return STRING;
  --CURSOR_NAME_FILE       : constant STRING := "ADASQL$ENV:CURSOR_DEFINITION";
  CURSOR_NAME            : constant STRING := "CURSOR_DEFINITION";
  CURSOR_NAME_ADA_SQL    : constant STRING := "CURSOR_DEFINITION";

-- database_name_file is as the file name should be accessed, without extention
-- database_name is the package name
-- database_name_ada_sql is the nexted package name

  function DATABASE_NAME_FILE return STRING;
  --DATABASE_NAME_FILE     : constant STRING := "ADASQL$ENV:DATABASE";
  DATABASE_NAME          : constant STRING := "DATABASE";
  DATABASE_NAME_ADA_SQL : constant STRING := "DATABASE";

-- dot_ada is the extention to be used with the files

  --DOT_ADA                : constant STRING := ".ADA";
  DOT_ADA_LEN            : constant POSITIVE := 4;
  DOT_ADA_UPPER          : constant STRING := ".ADA";
  DOT_ADA_LOWER          : constant STRING := ".ada";
  DOT_ADA_DEFAULT        : STRING (1..DOT_ADA_LEN) := ".ADA";

-- how_to_do_files  - if upper_case all file names are converted to upper case
--                    if lower_case all file names are converted to lower case
--                    if as_is they are to be used as entered by the user

  --HOW_TO_DO_FILES        : constant HOW_TO_DO_FILES_TYPE := UPPER_CASE;
  HOW_TO_DO_FILES        : HOW_TO_DO_FILES_TYPE;
  WHERE_IS_SCHEMA_FROM   : SCHEMA_FROM := UNKNOWN;
  SCHEMA_UNIT_CALLED     : STRING (1..200) := (others => ' ');
  SCHEMA_UNIT_CALLED_LEN : NATURAL := 0;

end IO_DEFINITIONS;
```

3.11.10 package ddl_io_defs.ada

```
package body IO_DEFINITIONS is
```

```
-- standard_name_file is as the file name should be accessed, without extention

  function STANDARD_NAME_FILE return STRING is
  begin
    return "ADASQL$ENV:STANDARD";
  end STANDARD_NAME_FILE;

-- cursor_name_file is as the file name should be accessed, without extention

  function CURSOR_NAME_FILE return STRING is
  begin
    return "ADASQL$ENV:CURSOR_DEFINITION";
  end CURSOR_NAME_FILE;

-- database_name_file is as the file name should be accessed, without extention

  function DATABASE_NAME_FILE return STRING is
  begin
    return "ADASQL$ENV:DATABASE";
  end DATABASE_NAME_FILE;

end IO_DEFINITIONS;
```

3.11.11 package ddl_defs.ada

```
with DATABASE, IO_DEFINITIONS;
use  DATABASE, IO_DEFINITIONS;

package DDL_DEFINITIONS is

  type STATUS_SCHEMA is (PROCESSING, WITHING, DONE, NOTOPEN, NOTFOUND);

  type KIND_TYPE is (A_TYPE, A_SUBTYPE, A_DERIVED, A_COMPONENT, A_VARIABLE);

  type TYPE_TYPE is (REC_ORD, ENUMERATION, INT_EGER, FL_OAT, STR_ING);

  type YET_TO_DO_DESCRIPTOR;
  type ACCESS_YET_TO_DO_DESCRIPTOR is access YET_TO_DO_DESCRIPTOR;

  type SCHEMA_UNIT_DESCRIPTOR;
  type ACCESS_SCHEMA_UNIT_DESCRIPTOR is access SCHEMA_UNIT_DESCRIPTOR;

  type WITHED_UNIT_DESCRIPTOR;
  type ACCESS_WITHED_UNIT_DESCRIPTOR is access WITHED_UNIT_DESCRIPTOR;

  type USED_PACKAGE_DESCRIPTOR;
  type ACCESS_USED_PACKAGE_DESCRIPTOR is access USED_PACKAGE_DESCRIPTOR;

  type DECLARED_PACKAGE_DESCRIPTOR;
  type ACCESS_DECLARED_PACKAGE_DESCRIPTOR is access
```

```
                                       DECLARED_PACKAGE_DESCRIPTOR;

type IDENTIFIER_DESCRIPTOR;
type ACCESS_IDENTIFIER_DESCRIPTOR is access IDENTIFIER_DESCRIPTOR;

type FULL_NAME_DESCRIPTOR;
type ACCESS_FULL_NAME_DESCRIPTOR is access FULL_NAME_DESCRIPTOR;

type TYPE_DESCRIPTOR(TY_PE : TYPE_TYPE);
type ACCESS_TYPE_DESCRIPTOR is access TYPE_DESCRIPTOR;

type LITERAL_DESCRIPTOR;
type ACCESS_LITERAL_DESCRIPTOR is access LITERAL_DESCRIPTOR;

type ENUM_LIT_DESCRIPTOR;
type ACCESS_ENUM_LIT_DESCRIPTOR is access ENUM_LIT_DESCRIPTOR;

type FULL_ENUM_LIT_DESCRIPTOR;
type ACCESS_FULL_ENUM_LIT_DESCRIPTOR is access FULL_ENUM_LIT_DESCRIPTOR;

type ENUM_LIT_NAME_STRING is new STRING;
type ENUM_LIT_NAME    is access ENUM_LIT_NAME_STRING;

type AUTH_IDENT_NAME_STRING is new STRING;
type AUTH_IDENT_NAME        is access AUTH_IDENT_NAME_STRING;

type LIBRARY_UNIT_NAME_STRING is new STRING;
type LIBRARY_UNIT_NAME        is access LIBRARY_UNIT_NAME_STRING;

type PACKAGE_NAME_STRING is new STRING;
type PACKAGE_NAME        is access PACKAGE_NAME_STRING;

type RECORD_NAME_STRING is new STRING;
type RECORD_NAME        is access RECORD_NAME_STRING;

type TYPE_NAME_STRING is new STRING;
type TYPE_NAME  is access TYPE_NAME_STRING;

type ENUMERATION_NAME_STRING is new STRING;
type ENUMERATION_NAME        is access ENUMERATION_NAME_STRING;

subtype ACCESS_RECORD_DESCRIPTOR  is ACCESS_TYPE_DESCRIPTOR(REC_ORD);
subtype ACCESS_ENUMERATION_DESCRIPTOR is ACCESS_TYPE_DESCRIPTOR(ENUMERATION);
subtype ACCESS_INTEGER_DESCRIPTOR is ACCESS_TYPE_DESCRIPTOR(INT_EGER);
subtype ACCESS_FLOAT_DESCRIPTOR   is ACCESS_TYPE_DESCRIPTOR(FL_OAT);
subtype ACCESS_STRING_DESCRIPTOR  is ACCESS_TYPE_DESCRIPTOR(STR_ING);

type YET_TO_DO_DESCRIPTOR is
  record
```

```
      UNDONE_SCHEMA                   : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
      PREVIOUS_YET_TO_DO              : ACCESS_YET_TO_DO_DESCRIPTOR;
      NEXT_YET_TO_DO                  : ACCESS_YET_TO_DO_DESCRIPTOR;
    end record;

  type SCHEMA_UNIT_DESCRIPTOR is
    record
      NAME                            : LIBRARY_UNIT_NAME;
      AUTH_ID                         : AUTH_IDENT_NAME;
      IS_AUTH_PACKAGE                 : BOOLEAN;
      HAS_DECLARED_TYPES              : BOOLEAN;
      HAS_DECLARED_TABLES             : BOOLEAN;
      HAS_DECLARED_VARIABLES          : BOOLEAN;
      FIRST_WITHED                    : ACCESS_WITHED_UNIT_DESCRIPTOR;
      LAST_WITHED                     : ACCESS_WITHED_UNIT_DESCRIPTOR;
      FIRST_USED                      : ACCESS_USED_PACKAGE_DESCRIPTOR;
      LAST_USED                       : ACCESS_USED_PACKAGE_DESCRIPTOR;
      FIRST_DECLARED_PACKAGE          : ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
      LAST_DECLARED_PACKAGE           : ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
      STREAM                          : INPUT_STREAM;
      SCHEMA_STATUS                   : STATUS_SCHEMA;
      PREVIOUS_SCHEMA_UNIT            : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
      NEXT_SCHEMA_UNIT                : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
    end record;

  type WITHED_UNIT_DESCRIPTOR is
    record
      SCHEMA_UNIT                     : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
      PREVIOUS_WITHED                 : ACCESS_WITHED_UNIT_DESCRIPTOR;
      NEXT_WITHED                     : ACCESS_WITHED_UNIT_DESCRIPTOR;
    end record;

  type USED_PACKAGE_DESCRIPTOR is
    record
      NAME                            : PACKAGE_NAME;
      PREVIOUS_USED                   : ACCESS_USED_PACKAGE_DESCRIPTOR;
      NEXT_USED                       : ACCESS_USED_PACKAGE_DESCRIPTOR;
    end record;

  type DECLARED_PACKAGE_DESCRIPTOR is
    record
      NAME                            : PACKAGE_NAME;
      FOUND_END                       : BOOLEAN;
      PREVIOUS_DECLARED               : ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
      NEXT_DECLARED                   : ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
    end record;

  type IDENTIFIER_DESCRIPTOR is
    record
```

```
      NAME              : TYPE_NAME;
      FIRST_FULL_NAME   : ACCESS_FULL_NAME_DESCRIPTOR;
      LAST_FULL_NAME    : ACCESS_FULL_NAME_DESCRIPTOR;
      PREVIOUS_IDENT    : ACCESS_IDENTIFIER_DESCRIPTOR;
      NEXT_IDENT        : ACCESS_IDENTIFIER_DESCRIPTOR;
    end record;

  type FULL_NAME_DESCRIPTOR is
    record
      NAME                 : TYPE_NAME;
      FULL_PACKAGE_NAME    : PACKAGE_NAME;
      TABLE_NAME           : RECORD_NAME;     -- null if not component
      IS_NOT_NULL          : BOOLEAN;         -- if it's suffixed onto the name
      IS_NOT_NULL_UNIQUE   : BOOLEAN;
      TYPE_IS              : ACCESS_TYPE_DESCRIPTOR;
      SCHEMA_UNIT          : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
      PREVIOUS_NAME        : ACCESS_FULL_NAME_DESCRIPTOR;
      NEXT_NAME            : ACCESS_FULL_NAME_DESCRIPTOR;
    end record;

  type TYPE_DESCRIPTOR(TY_PE : TYPE_TYPE) is
    record
      TYPE_KIND         : KIND_TYPE;  -- type, subtype, derived, component,
                                      -- variable
      WHICH_TYPE        : TYPE_TYPE;  -- record, enumeration, integer,
                                      -- float, string
      FULL_NAME         : ACCESS_FULL_NAME_DESCRIPTOR;
      NOT_NULL          : BOOLEAN;   -- if it's got the trait, original or
      NOT_NULL_UNIQUE   : BOOLEAN;   -- inherited
      FIRST_SUBTYPE     : ACCESS_TYPE_DESCRIPTOR; -- points to our children
                                                  -- subtypes
      LAST_SUBTYPE      : ACCESS_TYPE_DESCRIPTOR;
      FIRST_DERIVED     : ACCESS_TYPE_DESCRIPTOR; -- points to our children
                                                  -- derives
      LAST_DERIVED      : ACCESS_TYPE_DESCRIPTOR;
      FIRST_COMPONENT   : ACCESS_TYPE_DESCRIPTOR; -- points to our components
                                                  -- only if we're a record
                                                  -- type
      LAST_COMPONENT    : ACCESS_TYPE_DESCRIPTOR;
      PREVIOUS_ONE      : ACCESS_TYPE_DESCRIPTOR; -- chain of subtypes, derives,
                                                  -- or components from previous
                                                  -- six pointers, or chain of
                                                  -- tables if type-record
      NEXT_ONE          : ACCESS_TYPE_DESCRIPTOR;
      PREVIOUS_TYPE     : ACCESS_TYPE_DESCRIPTOR; -- chain of all types or
                                                  -- variables
      NEXT_TYPE         : ACCESS_TYPE_DESCRIPTOR;
      ULT_PARENT_TYPE   : ACCESS_TYPE_DESCRIPTOR; -- if there are no deriveds in
                                                  -- chain to this item it will
```

```
                                              -- be the same as base_type
                                              -- if there are deriveds
                                              -- it will be the ultimate
                                              -- parent of the derived
PARENT_TYPE       : ACCESS_TYPE_DESCRIPTOR;   -- if we're subtype, derived,
                                              -- component or variable this
                                              -- is our subtype-indicator
BASE_TYPE         : ACCESS_TYPE_DESCRIPTOR;   -- if we're subtype, derived,
                                              -- component or variable
PARENT_RECORD     : ACCESS_TYPE_DESCRIPTOR;   -- if we're component

case TY_PE is

when REC_ORD =>
    null;

when ENUMERATION =>
  FIRST_LITERAL       : ACCESS_LITERAL_DESCRIPTOR; -- if we're a type
                           -- this chain is all literals associated
                           -- with this enumeration, if we're a
                           -- subtype, derived, or component this
                           -- chain may be pointing to a partial
                           -- set of the parents chain
  LAST_LITERAL          : ACCESS_LITERAL_DESCRIPTOR;
  LAST_POS        : NATURAL; -- number of entries
  MAX_LENGTH            : NATURAL; -- max width of a field

when INT_EGER =>
    RANGE_LO_INT        : INT;
    RANGE_HI_INT        : INT;

when FL_OAT =>
    FLOAT_DIGITS        : NATURAL;
    RANGE_LO_FLT        : DOUBLE_PRECISION;
    RANGE_HI_FLT        : DOUBLE_PRECISION;

when STR_ING =>
  LENGTH              : NATURAL; -- 0 = not set or unconstrained
    INDEX_TYPE          : ACCESS_TYPE_DESCRIPTOR; -- points to the type of
                            -- item used to index array, optional
    ARRAY_TYPE          : ACCESS_TYPE_DESCRIPTOR; -- points to the type
                            -- of components in the array, which
                            -- ultimately must be character
    CONSTRAINED         : BOOLEAN;
    ARRAY_RANGE_LO      : INT;   --  -1 = not set (unconstrained)
    ARRAY_RANGE_HI      : INT;   --  -1 = not set (unconstrained)
    ARRAY_RANGE_MIN     : INT;   --  limits from parent or -1 =
    ARRAY_RANGE_MAX     : INT;   --  unconstrained or no limits placed
```

```
      end case;
    end record;

  type LITERAL_DESCRIPTOR is
    record
      NAME              : ENUMERATION_NAME;
      POS               : NATURAL;
      PARENT_ENUM           : ACCESS_TYPE_DESCRIPTOR;
      PREVIOUS_LITERAL      : ACCESS_LITERAL_DESCRIPTOR;
      NEXT_LITERAL          : ACCESS_LITERAL_DESCRIPTOR;
    end record;

  type ENUM_LIT_DESCRIPTOR is
    record
      NAME                    : ENUM_LIT_NAME;
      FIRST_FULL_ENUM_LIT     : ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
      LAST_FULL_ENUM_LIT      : ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
      PREVIOUS_ENUM_LIT       : ACCESS_ENUM_LIT_DESCRIPTOR;
      NEXT_ENUM_LIT           : ACCESS_ENUM_LIT_DESCRIPTOR;
    end record;

  type FULL_ENUM_LIT_DESCRIPTOR is
    record
      NAME                   : ENUM_LIT_NAME;
      TYPE_IS                : ACCESS_TYPE_DESCRIPTOR;
      PREVIOUS_LIT           : ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
      NEXT_LIT               : ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
    end record;

end DDL_DEFINITIONS;
```

3.11.12 package ddl_new_des_spec.ada

```
with DATABASE, DDL_DEFINITIONS;
use  DATABASE, DDL_DEFINITIONS;

package GET_NEW_DESCRIPTOR_ROUTINES is

  function GET_NEW_YET_TO_DO_DESCRIPTOR
           return ACCESS_YET_TO_DO_DESCRIPTOR;

  function GET_NEW_SCHEMA_UNIT_DESCRIPTOR
           return ACCESS_SCHEMA_UNIT_DESCRIPTOR;

  function GET_NEW_WITHED_UNIT_DESCRIPTOR
           return ACCESS_WITHED_UNIT_DESCRIPTOR;

  function GET_NEW_USED_PACKAGE_DESCRIPTOR
           return ACCESS_USED_PACKAGE_DESCRIPTOR;
```

```
function GET_NEW_DECLARED_PACKAGE_DESCRIPTOR
         return ACCESS_DECLARED_PACKAGE_DESCRIPTOR;

function GET_NEW_IDENTIFIER_DESCRIPTOR
         return ACCESS_IDENTIFIER_DESCRIPTOR;

function GET_NEW_FULL_NAME_DESCRIPTOR
         return ACCESS_FULL_NAME_DESCRIPTOR;

function GET_NEW_RECORD_DESCRIPTOR
         return ACCESS_RECORD_DESCRIPTOR;

function GET_NEW_ENUMERATION_DESCRIPTOR
         return ACCESS_ENUMERATION_DESCRIPTOR;

function GET_NEW_INTEGER_DESCRIPTOR
         return ACCESS_INTEGER_DESCRIPTOR;

function GET_NEW_FLOAT_DESCRIPTOR
         return ACCESS_FLOAT_DESCRIPTOR;

function GET_NEW_STRING_DESCRIPTOR
         return ACCESS_STRING_DESCRIPTOR;

function GET_NEW_TYPE_DESCRIPTOR
        (IN_TYPE     : in TYPE_TYPE)
         return ACCESS_TYPE_DESCRIPTOR;

function GET_NEW_LITERAL_DESCRIPTOR
         return ACCESS_LITERAL_DESCRIPTOR;

function GET_NEW_ENUM_LIT_DESCRIPTOR
         return ACCESS_ENUM_LIT_DESCRIPTOR;

function GET_NEW_FULL_ENUM_LIT_DESCRIPTOR
         return ACCESS_FULL_ENUM_LIT_DESCRIPTOR;

function GET_NEW_ENUM_LIT_NAME
        (TEMP : in STRING)
         return ENUM_LIT_NAME;

function GET_NEW_AUTH_IDENT_NAME
        (TEMP : in STRING)
         return AUTH_IDENT_NAME;

function GET_NEW_LIBRARY_UNIT_NAME
        (TEMP : in STRING)
         return LIBRARY_UNIT_NAME;
```

```
  function GET_NEW_PACKAGE_NAME
          (TEMP : in STRING)
           return PACKAGE_NAME;

  function GET_NEW_RECORD_NAME
          (TEMP : in STRING)
           return RECORD_NAME;

  function GET_NEW_TYPE_NAME
          (TEMP : in STRING)
           return TYPE_NAME;

  function GET_NEW_ENUMERATION_NAME
          (TEMP : in STRING)
           return ENUMERATION_NAME;

end GET_NEW_DESCRIPTOR_ROUTINES;
```

3.11.13 package ddl_extra_defs.ada

```
with DATABASE, DDL_DEFINITIONS;
use  DATABASE, DDL_DEFINITIONS;

package EXTRA_DEFINITIONS is

  type PROCESS_TYPE is (ITS_WITH, ITS_ALREADY_WITHING, ITS_USE, ITS_PACKAGE,
                        ITS_END, ITS_TYPE, ITS_SUBTYPE, ITS_FUNCTION,
                        ITS_SCHEMA_AUTHORIZATION, ITS_EOL, ITS_UNKNOWN,
                        ITS_FINISHED);

  type NAME_TO_PROCESS_LIST;
  type ACCESS_NAME_TO_PROCESS_LIST is access NAME_TO_PROCESS_LIST;

  type LIST_NAME_STRING is new STRING;
  type LIST_NAME        is access LIST_NAME_STRING;

  type NAME_TO_PROCESS_LIST is
    record
      NAME                : LIST_NAME;
      PREVIOUS_NAME       : ACCESS_NAME_TO_PROCESS_LIST;
      NEXT_NAME           : ACCESS_NAME_TO_PROCESS_LIST;
    end record;

  type COMPONENT_TO_PROCESS_LIST;
  type ACCESS_COMPONENT_TO_PROCESS_LIST is access COMPONENT_TO_PROCESS_LIST;

  type LIST_COMPONENT_STRING is new STRING;
  type LIST_COMPONENT        is access LIST_COMPONENT_STRING;

  type COMPONENT_TO_PROCESS_LIST is
```

```
  record
    COMPONENT               : LIST_COMPONENT;
    PREVIOUS_COMPONENT      : ACCESS_COMPONENT_TO_PROCESS_LIST;
    NEXT_COMPONENT          : ACCESS_COMPONENT_TO_PROCESS_LIST;
  end record;

type HOLDING_COMPONENT_DESCRIPTOR;
type ACCESS_HOLDING_COMPONENT_DESCRIPTOR is access
            HOLDING_COMPONENT_DESCRIPTOR;

type HOLDING_COMPONENT_DESCRIPTOR is
  record
    WHICH_TYPE           : TYPE_TYPE;
    FULL_NAME            : TYPE_NAME := null;
    ULT_PARENT_TYPE      : ACCESS_TYPE_DESCRIPTOR := null;
    PARENT_TYPE          : ACCESS_TYPE_DESCRIPTOR := null;
    BASE_TYPE            : ACCESS_TYPE_DESCRIPTOR := null;
    PREVIOUS_COMPONENT   : ACCESS_HOLDING_COMPONENT_DESCRIPTOR := null;
    NEXT_COMPONENT       : ACCESS_HOLDING_COMPONENT_DESCRIPTOR := null;
    NOT_NULL             : BOOLEAN := FALSE;
    NOT_NULL_UNIQUE      : BOOLEAN := FALSE;
    FIRST_LITERAL        : ACCESS_LITERAL_DESCRIPTOR := null;
    LAST_LITERAL         : ACCESS_LITERAL_DESCRIPTOR := null;
    LAST_POS             : NATURAL := 0;
    MAX_LENGTH           : NATURAL := 0;
    RANGE_LO_INT         : INT := 0;
    RANGE_HI_INT         : INT := 0;
    FLOAT_DIGITS         : NATURAL := 0;
    RANGE_LO_FLT         : DOUBLE_PRECISION := 0.0;
    RANGE_HI_FLT         : DOUBLE_PRECISION := 0.0;
    LENGTH               : NATURAL := 0;
    INDEX_TYPE           : ACCESS_TYPE_DESCRIPTOR := null;
    ARRAY_TYPE           : ACCESS_TYPE_DESCRIPTOR := null;
    CONSTRAINED          : BOOLEAN := TRUE;
    ARRAY_RANGE_LO       : INT := 0;
    ARRAY_RANGE_HI       : INT := 0;
    ARRAY_RANGE_MIN      : INT := 0;
    ARRAY_RANGE_MAX      : INT := 0;
  end record;

CURRENT_SCHEMA_UNIT        : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
CURRENT_PROCESS            : PROCESS_TYPE := ITS_UNKNOWN;
AUTH_ID                    : AUTH_IDENT_NAME := null;
OUR_PACKAGE_NAME           : STRING (1..250);
OUR_PACKAGE_NAME_LAST      : NATURAL := 0;
ADA_NAME                   : STRING (1..250);
ADA_NAME_LAST              : NATURAL := 0;
TEMP_STRING                : STRING (1..250);
TEMP_STRING_LAST           : NATURAL := 0;
```

```
  FIRST_NAME_TO_PROCESS       : ACCESS_NAME_TO_PROCESS_LIST := null;
  LAST_NAME_TO_PROCESS        : ACCESS_NAME_TO_PROCESS_LIST := null;
  FIRST_COMPONENT_TO_PROCESS : ACCESS_COMPONENT_TO_PROCESS_LIST := null;
  LAST_COMPONENT_TO_PROCESS  : ACCESS_COMPONENT_TO_PROCESS_LIST := null;
  DEBUGGING                   : BOOLEAN := FALSE;
  SCHEMA_DEF_NAME             : constant STRING := "SCHEMA_DEFINITION";
  ADA_SQL_PACK                : constant STRING := "ADA_SQL";
  SUF_NOT_NULL                : constant STRING := "_NOT_NULL";
  SUF_NOT_NULL_LEN            : constant INTEGER := 9;
  SUF_UNIQUE                  : constant STRING := "_NOT_NULL_UNIQUE";
  SUF_UNIQUE_LEN              : constant INTEGER := 16;
  CHARACTER_BASE              : constant STRING := "CHARACTER";

end EXTRA_DEFINITIONS;
```

3.11.14 package enums.ada

```
-- enums.ada -- manage internal data strucs for enumeration type overloading

with DDL_DEFINITIONS;
package ENUMERATION is

-- Because of Ada overloading, when we encounter an enumeration literal we do
-- not necessarily know of what type it is.  Instead, we have a list of
-- possible types that (1) are visible, and (2) have the given literal as a
-- value.

-- Each entry in the list of possible types is very simple -- it merely
-- indicates which type the entry represents.  All required type information
-- is present in an ACCESS_FULL_NAME_DESCRIPTOR, and this is the data
-- structure used by routines in this package to communicate type information.

-- Calling routines refer to a list of possible types via an object of type
-- ENUMERATION.TYPE_LIST.  The various routines provided here manipulate and
-- interrogate these lists.

  type TYPE_LIST is private;

-- ENUMERATION.TYPE_LIST_CREATOR returns a data structure representing an
-- empty list of possible enumeration types.  It is called when we have
-- encountered an enumeration literal, and will subsequently call
-- ENUMERATION.TYPE_GOES_ON_LIST (see below) to add all possible types to our
-- new list.

  function TYPE_LIST_CREATOR return TYPE_LIST;

-- ENUMERATION.TYPE_GOES_ON_LIST causes the indicated type to be added to the
-- given list of possible types.  It is called once for each possible type
-- determined for a particular enumeration literal.
```

```
  procedure TYPE_GOES_ON_LIST
            ( TYPE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
              LIST            : TYPE_LIST );

-- When processing an operator, it is possible that one operand is of a known
-- type (for example, a database column), while the other operand is an
-- overloaded enumeration literal.  In order to determine whether or not the
-- operation is valid, it is necessary to see if the known type appears on the
-- list of possible types for the overloaded literal.  This is the purpose of
-- ENUMERATION.TYPE_IS_ON_LIST, which returns TRUE if the indicated type is
-- on the given list, FALSE otherwise.

  function TYPE_IS_ON_LIST
           ( TYPE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
             LIST            : TYPE_LIST ) return BOOLEAN;

-- There are certain pathological constructs (e.g., enumeration_literal_1 <
-- enumeration_literal_2) wherein both operands of an operator can be
-- overloaded.  When processing the operator, the intersection of the possible
-- type lists for the two operands determines the set of possible operand
-- types.  ENUMERATION.TYPE_LIST_INTERSECTION returns a list of possible types
-- representing the intersection of its two given lists.

  function TYPE_LIST_INTERSECTION ( LEFT , RIGHT : TYPE_LIST )
   return TYPE_LIST;

-- When two type lists are intersected to determine operand types for an
-- operator, there are three possible results of interest:
--
-- (1) If the intersection contains no possible types, then the operation is
--     not valid
--
-- (2) If the intersection contains one possible type, then the operation has
--     been uniquely determined
--
-- (3) If the intersection contains more than one possible type, then the
--     operation has not been uniquely determined.  This is an error for
--     typical binary operations, such as enumeration_literal_1 <
--     enumeration_literal_2.  There is, however, at least one REALLY
--     pathological construct, containing a list of operators, where it may
--     be necessary to consider several subsequent intersections before the
--     operations can be uniquely determined.  An example is:
--
--        IS_IN ( enumeration_literal_1,
--                enumeration_literal_2 OR enumeration_literal_3 ... );

-- ENUMERATION.TYPE_LIST_SIZE returns the number of entries in the given list
-- of possible types, except that 2 represents any number greater than 1.
-- (The languages of some primitive cultures supposedly incorporate similar
```

```
-- ideas.)

  subtype ZERO_ONE_MANY is INTEGER range 0 .. 2;

  function TYPE_LIST_SIZE ( LIST : TYPE_LIST ) return ZERO_ONE_MANY;

-- If we are fortunate enough to intersect two possible type lists down to a
-- single type, we will ultimately have to know what type it is.
-- ENUMERATION.TYPE_ON_LIST returns the ACCESS_FULL_NAME_DESCRIPTOR of the
-- (presumably) only type on the given list.

  function TYPE_ON_LIST ( LIST : TYPE_LIST )
   return DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;

private

  type TYPE_LIST_RECORD;

  type TYPE_LIST is access TYPE_LIST_RECORD;

  type TYPE_LIST_RECORD is
    record
      DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
      NEXT_TYPE  : TYPE_LIST;
    end record;

end ENUMERATION;
```

3.11.15 package enumb.ada

```
-- enumb.ada -- manage internal data structures for enum type overloading

with DDL_DEFINITIONS;
package body ENUMERATION is

   use DDL_DEFINITIONS;

function TYPE_LIST_CREATOR
   return TYPE_LIST is
begin
   return new TYPE_LIST_RECORD'(DESCRIPTOR => null, NEXT_TYPE => null);
end TYPE_LIST_CREATOR;

procedure TYPE_GOES_ON_LIST
   (TYPE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    LIST            : TYPE_LIST) is
begin
   if LIST.DESCRIPTOR = null then
      LIST.DESCRIPTOR := TYPE_DESCRIPTOR;
   else
```

```
      LIST.NEXT_TYPE := new TYPE_LIST_RECORD'
                                  (DESCRIPTOR => TYPE_DESCRIPTOR,
                                   NEXT_TYPE  => LIST.NEXT_TYPE);
   end if;
end TYPE_GOES_ON_LIST;

function TYPE_IS_ON_LIST
   (TYPE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    LIST            : TYPE_LIST)
   return BOOLEAN is
   TRACER : TYPE_LIST := LIST;
begin
   while TRACER /= null and then
      TRACER.DESCRIPTOR /= TYPE_DESCRIPTOR loop
      TRACER := TRACER.NEXT_TYPE;
   end loop;
   if TRACER /= null then
      return TRUE;
   else
      return FALSE;
   end if;
end TYPE_IS_ON_LIST;

function TYPE_LIST_INTERSECTION
   (LEFT, RIGHT : TYPE_LIST)
   return TYPE_LIST is
   RESULT : TYPE_LIST := TYPE_LIST_CREATOR;
   TRACER : TYPE_LIST := LEFT;
begin
   while TRACER /= null and then
      TRACER.DESCRIPTOR /= null loop
      if TYPE_IS_ON_LIST (TRACER.DESCRIPTOR, RIGHT) then
         TYPE_GOES_ON_LIST (TRACER.DESCRIPTOR, RESULT);
      end if;
      TRACER := TRACER.NEXT_TYPE;
   end loop;
   return RESULT;
end TYPE_LIST_INTERSECTION;

function TYPE_LIST_SIZE
   (LIST : TYPE_LIST)
   return ZERO_ONE_MANY is
begin
   if LIST.DESCRIPTOR = null then
      return 0;
   elsif LIST.NEXT_TYPE = null then
      return 1;
   else
      return 2;
```

```
   end if;
end TYPE_LIST_SIZE;

function TYPE_ON_LIST
   (LIST : TYPE_LIST)
   return DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR is
begin
   return LIST.DESCRIPTOR;
end TYPE_ON_LIST;

end ENUMERATION;
```

3.11.16 package dummys.ada

```
-- dummys.ada - dummy data structure entries with null strings for lists

with DDL_DEFINITIONS;
package DUMMY is

  LIBRARY_UNIT_NAME : constant DDL_DEFINITIONS.LIBRARY_UNIT_NAME :=
   new DDL_DEFINITIONS.LIBRARY_UNIT_NAME_STRING' ( "" );

  TYPE_NAME : constant DDL_DEFINITIONS.TYPE_NAME :=
   new DDL_DEFINITIONS.TYPE_NAME_STRING' ( "" );

  RECORD_NAME : constant DDL_DEFINITIONS.RECORD_NAME :=
   new DDL_DEFINITIONS.RECORD_NAME_STRING' ( "" );
  ACCESS_SCHEMA_UNIT_DESCRIPTOR :
   constant DDL_DEFINITIONS.ACCESS_SCHEMA_UNIT_DESCRIPTOR :=
    new DDL_DEFINITIONS.SCHEMA_UNIT_DESCRIPTOR'
        ( LIBRARY_UNIT_NAME, null, FALSE, FALSE, FALSE, FALSE, null, null,
          null, null, null, null, null, DDL_DEFINITIONS.DONE, null, null );

  ACCESS_FULL_NAME_DESCRIPTOR :
   constant DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
    new DDL_DEFINITIONS.FULL_NAME_DESCRIPTOR'
        ( TYPE_NAME, null, null, FALSE, FALSE, null,
          ACCESS_SCHEMA_UNIT_DESCRIPTOR, null, null );

  ACCESS_TYPE_DESCRIPTOR : constant DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
   new DDL_DEFINITIONS.TYPE_DESCRIPTOR'
       ( DDL_DEFINITIONS.REC_ORD,
         DDL_DEFINITIONS.A_TYPE,
         DDL_DEFINITIONS.REC_ORD,
         ACCESS_FULL_NAME_DESCRIPTOR,
         FALSE, FALSE,
         null, null, null, null, null, null, null,
         null, null, null, null, null, null, null );

end DUMMY;
```

3.11.17 package ddl_variables.ada

```
with DDL_DEFINITIONS;
use DDL_DEFINITIONS;

package DDL_VARIABLES is

  FIRST_YET_TO_DO      : ACCESS_YET_TO_DO_DESCRIPTOR := null;
  LAST_YET_TO_DO       : ACCESS_YET_TO_DO_DESCRIPTOR := null;
  FIRST_SCHEMA_UNIT    : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
  LAST_SCHEMA_UNIT     : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
  FIRST_IDENTIFIER     : ACCESS_IDENTIFIER_DESCRIPTOR := null;
  LAST_IDENTIFIER      : ACCESS_IDENTIFIER_DESCRIPTOR := null;
  FIRST_TYPE           : ACCESS_TYPE_DESCRIPTOR := null;
  LAST_TYPE            : ACCESS_TYPE_DESCRIPTOR := null;
  FIRST_TABLE          : ACCESS_TYPE_DESCRIPTOR := null;
  LAST_TABLE           : ACCESS_TYPE_DESCRIPTOR := null;
  FIRST_VARIABLE       : ACCESS_TYPE_DESCRIPTOR := null;
  LAST_VARIABLE        : ACCESS_TYPE_DESCRIPTOR := null;
  FIRST_ENUM_LIT       : ACCESS_ENUM_LIT_DESCRIPTOR := null;
  LAST_ENUM_LIT        : ACCESS_ENUM_LIT_DESCRIPTOR := null;

end DDL_VARIABLES;
```

3.11.18 package columns.ada

```
with DDL_DEFINITIONS, LEXICAL_ANALYZER;

package COLUMN_LIST is

  type ELEMENT_RECORD;
  type ELEMENT is access ELEMENT_RECORD;

  type ELEMENT_RECORD is
    record
      COLUMN_DES  : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
      NEXT_COLUMN : ELEMENT := null;
    end record;

  procedure ADD_NEW_COLUMN
           (CURRENT_LIST : in out ELEMENT;
            ADD_COLUMN   : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
            TOKEN        : LEXICAL_ANALYZER.LEXICAL_TOKEN);

end COLUMN_LIST;
```

3.11.19 package columnb.ada

```
with DDL_DEFINITIONS;
use  DDL_DEFINITIONS;
```

```
package body COLUMN_LIST is

  procedure ADD_NEW_COLUMN
           (CURRENT_LIST : in out ELEMENT;
            ADD_COLUMN   : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
            TOKEN        : LEXICAL_ANALYZER.LEXICAL_TOKEN) is

    LIST       : ELEMENT;
    DUPLICATED : BOOLEAN := FALSE;

  begin
    if CURRENT_LIST = null then
      CURRENT_LIST := new ELEMENT_RECORD '
         (COLUMN_DES   => ADD_COLUMN,
          NEXT_COLUMN  => null),
      return;
    end if;
    LIST := CURRENT_LIST;
    while LIST.NEXT_COLUMN /= null loop
      if ADD_COLUMN = LIST.COLUMN_DES then
        DUPLICATED := TRUE;
      end if;
      LIST := LIST.NEXT_COLUMN;
    end loop;
    LIST.NEXT_COLUMN := new ELEMENT_RECORD '
       (COLUMN_DES   => ADD_COLUMN,
        NEXT_COLUMN  => null);
    if DUPLICATED then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, "Duplicated column name");
    end if;
  end ADD_NEW_COLUMN;
end COLUMN_LIST;
```

3.11.20 package withs.ada

```
-- withs.ada - post process data structures for library units to be with'ed

package WITH_REQUIRED is

-- The code generated by the application scanner with's all units mentioned in
-- that section of the application program's context clause that we process
-- (first with in this implementation).  This is done to force reprocessing of
-- an application program if any of its with'ed units changes.  (It is also
-- required for those units that we reference.)  Of course, Ada only forces
-- recompilation of the generated unit; the programmer will hopefully remember
-- to regenerate (as opposed to degenerate) as well.

-- The context clause we generate looks like:
--
--   with ADA_SQL_FUNCTIONS, DATABASE, x, y, ...;
```

```
--
-- where x, y, ... are the names of library units with'ed by the application
-- program.  DATABASE may not really be required, but we with it anyway, for
-- simplicity, rather than having to figure out if it is required.

-- The DDL reader data structures contain the list of library units that the
-- application program with'ed, so we do not need any special data structure
-- to record the information.  When post processing from the DDL reader data
-- structures to produce the context clause, we do not repeat DATABASE if it
-- also appears in the application program context clause (for neatness).
-- STANDARD will also appear in the DDL reader data structures, guaranteed to
-- be the first entry, but does not appear in the generated context clause.

-- The procedure to produce the generated context clause is:

  procedure POST_PROCESSING;

end WITH_REQUIRED;
```

3.11.21 package withb.ada

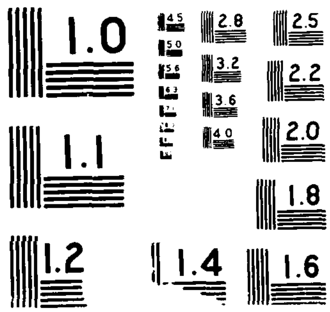
```
-- withb.ada - post process data structures for library units to be with'ed

with TEXT_PRINT, DDL_DEFINITIONS, EXTRA_DEFINITIONS;
use  TEXT_PRINT;
package body WITH_REQUIRED is

   use DDL_DEFINITIONS;

   COMPILATION_UNIT_BEING_SCANNED
      : DDL_DEFINITIONS.ACCESS_SCHEMA_UNIT_DESCRIPTOR renames
        EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT;

procedure POST_PROCESSING is
   TRACER : DDL_DEFINITIONS.ACCESS_WITHED_ NIT_DESCRIPTOR :=
      COMPILATION_UNIT_BEING_SCANNED.FIRST_WITHED;
begin
   SET_INDENT (0);
   PRINT ("with ADA_SQL_FUNCTIONS, DATABASE");
   while TRACER /= null loop
      if STRING(TRACER.SCHEMA_UNIT.NAME.all) /= "STANDARD" and then
         STRING(TRACER.SCHEMA_UNIT.NAME.all) /= "DATABASE" then
         PRINT (", ");
         PRINT (STRING(TRACER.SCHEMA_UNIT.NAME.all));
      end if;
      TRACER := TRACER.NEXT_WITHED;
   end loop;
   PRINT (";");
   PRINT_LINE;
end POST_PROCESSING;
```

```
end WITH_REQUIRED;
```

3.11.22 package results.ada

```
-- results.ada - internal data struc for keeping track of function result type

with DDL_DEFINITIONS, ENUMERATION;
package RESULT is

-- As we scan through an Ada/SQL program, we process expressions, comprised of
-- program objects and database objects.  In order to know what subprograms we
-- must generate, we must keep track of the types of these expressions.

-- The result of an expression may be either of a program type (standard Ada,
-- we generate no functions for the expression) or of a database type (our
-- special types representing database objects, for which we do generate
-- functions).  Values of the following enumeration type indicate where the
-- result value logically resides:

  type VALUE_LOCATION is ( IN_PROGRAM , IN_DATABASE );

-- A program expression may be of a known type (if it contains variables, for
-- ex-mple), or of an unknown type (if it contains literals that may belong to
-- more than one type).  A database expression containing at least one column
-- name will be of a known type, since the column will be of a known type.  It
-- is not necessary that a database expression contain any column names,
-- however, since the result of an INDICATOR function is considered a database
-- value.  If the parameter to the INDICATOR function is of an unknown
-- (necessarily program) type, then the value of the INDICATOR function is
-- also of an unknown (database) type.  Values of the following enumeration
-- type indicate whether the type is known or unknown:

  type TYPE_KNOWLEDGE is ( IS_KNOWN , IS_UNKNOWN );

-- Even if the type of an expression is unknown, we still know to what class
-- the type belongs, based on the literals used to construct the expression.
-- Values of the following enumeration subtype indicate the class to which an
-- unknown value belongs:

  subtype TYPE_CLASS is DDL_DEFINITIONS.TYPE_TYPE
   range DDL_DEFINITIONS.ENUMERATION .. DDL_DEFINITIONS.STR_ING;

-- Character literals are in the enumeration class.

-- A value of one of the last three classes can be of any type declared within
-- the class.  This is not the case with enumeration types, however.  An
-- enumeration literal can only be of a type that declares it as a value.  For
-- an unknown enumeration type, therefore, we also store a list of types to
-- which the value can belong.  This is an ENUMERATION.TYPE_LIST (see enums.-
-- ada).  The following data structure contains information about unknown
```

AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION 2/6
SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA
B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317
UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

```
-- types:

  type UNKNOWN_TYPE_DESCRIPTOR
       ( CLASS : TYPE_CLASS := DDL_DEFINITIONS.INT_EGER ) is
    record
      case CLASS is
        when DDL_DEFINITIONS.INT_EGER | DDL_DEFINITIONS.FL_OAT |
             DDL_DEFINITIONS.STR_ING =>
          null;
        when DDL_DEFINITIONS.ENUMERATION =>
          POSSIBLE_TYPES : ENUMERATION.TYPE_LIST;
      end case;
    end record;

-- For a known type, we simply store a pointer to the ACCESS_TYPE_DESCRIPTOR
-- for it.  Consequently, our complete data structure for representing the
-- type of an expression is:

  type DESCRIPTOR ( TYPE_IS : TYPE_KNOWLEDGE := IS_KNOWN ) is
    record
      LOCATION : VALUE_LOCATION;
      case TYPE_IS is
        when IS_KNOWN =>
         KNOWN_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
        when IS_UNKNOWN =>
         UNKNOWN_TYPE : UNKNOWN_TYPE_DESCRIPTOR;
      end case;
    end record;

-- The result type of each source file expression we process determines the
-- parameter and result types of the subprograms we generate.  There are two
-- basic operations that we perform on these result types when we consider
-- binary operations:

-- (1) Check two types for comparability: If we see A op B, we apply Ada/SQL's
--     strong typing by verifying that the types of A and B are comparable.
--     Comparability where types may be unknown is defined as follows:
--       If both types are known, then
--         They must be the same
--       Else
--         Both types must be of the same class
--         If the class is enumeration, then
--           If one type is known, then
--             The known type must appear on the possible type list of the
--              unknown type
--           Else (neither type is known)
--             The intersection of the possible type lists for the two types
--              must not be null (i.e., there must be at least one type in
--              common between the two lists)
```

```
--          End If
--        End If
--      End If

-- (2) Find the "combined result type" of two types.  Having seen A op B, and
--      verified that A is comparable with B, we then want to know what the
--      result type of the operation is.  In summary, if the type of at least
--      one operand is known, then the type of the result of the operation is
--      known.  Likewise, if at least one operand is of a database type, then
--      we will have to generate the subprogram for the operator, and the
--      result is of a database type.  This is spelled out in greater detail
--      below (the program/database flag is considered independently of the
--      known/unknown flag and associated information):
--        If either type is known, then
--          The result type is known to be that of the known type (either type
--           if both types are known, since they are comparable)
--        Elsif enumeration types are involved, then
--          The possible type list of the result is given by the intersection
--           of the two operand possible type lists
--          Case size of resulting type list is
--            When 0 =>
--             (not possible, since types are comparable)
--            When 1 =>
--             the result type is known, as given by the one possible type
--              common to both operands
--            When others (>2) =>
--             the result type is unknown, of enumeration class
--          End Case
--        Else
--          The result type is unknown, of the same class as the operand types
--        End If
--        If either type is a database type, then
--          The result type is a database type
--        Else
--          The result type is a program type
--        End If

-- Functions (1) and (2) are combined into a single routine, taking two
-- operand types as arguments and returning a comparability flag and a
-- combined result type (which is valid only if the comparability flag =
-- RESULT.IS_COMPARABLE).  There are two flavors of the routine, one in which
-- both operand types are given as RESULT.DESCRIPTORs, and one in which one of
-- the operand types is given as an ACCESS_TYPE_DESCRIPTOR.

  type COMPARABILITY is ( IS_COMPARABLE , IS_NOT_COMPARABLE );

  procedure COMBINED_TYPE
            ( LEFT        : in  DESCRIPTOR;
              RIGHT       : in  DESCRIPTOR;
```

```
                RESULT     : out DESCRIPTOR;
                COMPARABLE : out COMPARABILITY );

  procedure COMBINED_TYPE
               ( LEFT      : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
                RIGHT      : in  DESCRIPTOR;
                RESULT     : out DESCRIPTOR;
                COMPARABLE : out COMPARABILITY );

end RESULT;
```

3.11.23 package resultb.ada

```
-- resultb.ada - internal data struc for keeping track of function result type

with DDL_DEFINITIONS, ENUMERATION;
package body RESULT is

   use DDL_DEFINITIONS;

function CLASS_OF
   (TYPE_DESCRIPTOR : DESCRIPTOR)
   return TYPE_CLASS is
begin
   if TYPE_DESCRIPTOR.TYPE_IS = IS_KNOWN then
      return TYPE_DESCRIPTOR.KNOWN_TYPE.WHICH_TYPE;
   else
      return TYPE_DESCRIPTOR.UNKNOWN_TYPE.CLASS;
   end if;
end CLASS_OF;

function IS_COMPARABLE
   (LEFT, RIGHT : DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.TYPE_IS = IS_KNOWN and RIGHT.TYPE_IS = IS_KNOWN then
      -- both types are known, they are comparable if they are the same type.
      return LEFT.KNOWN_TYPE.BASE_TYPE = RIGHT.KNOWN_TYPE.BASE_TYPE;
   elsif CLASS_OF (LEFT) = CLASS_OF (RIGHT) then
      -- both types are of the same class.
      if CLASS_OF (LEFT) = DDL_DEFINITIONS.ENUMERATION then
         -- both types are enumeration
         if LEFT.TYPE_IS = IS_KNOWN then
            -- left type is known, the types are comparable if the known type
            -- of left is on the list of possible types for the unknown right
            -- type.
            return ENUMERATION.TYPE_IS_ON_LIST
               (LEFT.KNOWN_TYPE.FULL_NAME, RIGHT.UNKNOWN_TYPE.POSSIBLE_TYPES);
         elsif RIGHT.TYPE_IS = IS_KNOWN then
            -- right type is known, the types are comparable if the known type
```

```
            -- of right is on the list of possible types for the unknown left
            -- type.
            return ENUMERATION.TYPE_IS_ON_LIST
               (RIGHT.KNOWN_TYPE.FULL_NAME, LEFT.UNKNOWN_TYPE.POSSIBLE_TYPES);
         else  -- neither type is known
            -- the types (both of which are enumeration) are comparable if the
            -- intersection of the possible types for the two unknown types is
            -- not null;
            return ENUMERATION.TYPE_LIST_SIZE
                      (ENUMERATION.TYPE_LIST_INTERSECTION
                          (LEFT.UNKNOWN_TYPE.POSSIBLE_TYPES,
                           RIGHT.UNKNOWN_TYPE.POSSIBLE_TYPES)) > 0;
         end if;
      else
        return TRUE;
      end if;
   end if;
   return FALSE;
end IS_COMPARABLE;

procedure COMBINE_TYPES
   (LEFT, RIGHT : in  DESCRIPTOR;
    RESULT      : out DESCRIPTOR) is
   RESULT_LOC : VALUE_LOCATION;
   -- assume types are comparable
begin
   if LEFT.LOCATION = IN_DATABASE or RIGHT.LOCATION = IN_DATABASE then
      RESULT_LOC := IN_DATABASE;
   else
      RESULT_LOC := IN_PROGRAM;
   end if;
   if LEFT.TYPE_IS = IS_KNOWN then
      RESULT := LEFT;
      RESULT.LOCATION := RESULT_LOC;
   elsif RIGHT.TYPE_IS = IS_KNOWN then
      RESULT := RIGHT;
      RESULT.LOCATION := RESULT_LOC;
   elsif CLASS_OF(LEFT) = DDL_DEFINITIONS.ENUMERATION then
      declare
         INTERSECTION : ENUMERATION.TYPE_LIST :=
            ENUMERATION.TYPE_LIST_INTERSECTION
               (LEFT.UNKNOWN_TYPE.POSSIBLE_TYPES,
                RIGHT.UNKNOWN_TYPE.POSSIBLE_TYPES);
      begin
         case ENUMERATION.TYPE_LIST_SIZE (INTERSECTION) is
            when 0 => null;  -- cannot happen
            when 1 =>
               RESULT := DESCRIPTOR'
                            (TYPE_IS    => IS_KNOWN,
```

```
                                 LOCATION    => RESULT_LOC,
                                 KNOWN_TYPE => ENUMERATION.TYPE_ON_LIST
                                                  (INTERSECTION).TYPE_IS);
                  when others =>
                     RESULT := DESCRIPTOR'
                                (TYPE_IS      => IS_UNKNOWN,
                                 LOCATION     => RESULT_LOC,
                                 UNKNOWN_TYPE =>
                                    UNKNOWN_TYPE_DESCRIPTOR'
                                       (CLASS          => DDL_DEFINITIONS.ENUMERATIO
                                        POSSIBLE_TYPES => INTERSECTION));
               end case;
            end;
         else
            declare
               UNKNOWN_DESCRIPTOR : UNKNOWN_TYPE_DESCRIPTOR;
            begin
               case CLASS_OF (LEFT) is
                  when DDL_DEFINITIONS.INT_EGER =>
                     UNKNOWN_DESCRIPTOR :=
                        UNKNOWN_TYPE_DESCRIPTOR'(CLASS => DDL_DEFINITIONS.INT_EGER);
                  when DDL_DEFINITIONS.FL_OAT =>
                     UNKNOWN_DESCRIPTOR :=
                        UNKNOWN_TYPE_DESCRIPTOR'(CLASS => DDL_DEFINITIONS.FL_OAT);
                  when DDL_DEFINITIONS.STR_ING =>
                     UNKNOWN_DESCRIPTOR :=
                        UNKNOWN_TYPE_DESCRIPTOR'(CLASS => DDL_DEFINITIONS.STR_ING);
                  when others => null;  -- can't happen
               end case;
               RESULT := DESCRIPTOR'
                          (TYPE_IS      => IS_UNKNOWN,
                           LOCATION     => RESULT_LOC,
                           UNKNOWN_TYPE => UNKNOWN_DESCRIPTOR);
            end;
         end if;
      end COMBINE_TYPES;

      procedure COMBINED_TYPE
         (LEFT        : in  DESCRIPTOR;
          RIGHT       : in  DESCRIPTOR;
          RESULT      : out DESCRIPTOR;
          COMPARABLE  : out COMPARABILITY) is
      begin
         if IS_COMPARABLE (LEFT, RIGHT) then
            COMPARABLE := IS_COMPARABLE;
            COMBINE_TYPES (LEFT, RIGHT, RESULT);
         else
            COMPARABLE := IS_NOT_COMPARABLE;
         end if;
```

```
end COMBINED_TYPE;

procedure COMBINED_TYPE
   (LEFT        : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    RIGHT       : in  DESCRIPTOR;
    RESULT      : out DESCRIPTOR;
    COMPARABLE  : out COMPARABILITY) is
begin
   COMBINED_TYPE
      (DESCRIPTOR'(TYPE_IS => IS_KNOWN, LOCATION => IN_PROGRAM, KNOWN_TYPE => LEFT),
       RIGHT,
       RESULT,
       COMPARABLE);
end COMBINED_TYPE;

end RESULT;
```

3.11.24 package indexs.ada

```
-- indexs.ada - post process data strucs for generated index subtypes needed

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package INDEX_SUBTYPE is

-- In Ada, the subtype of the index of a constrained array may be anonymous,
-- examples:
--
--   type DEPT_NAME    is new STRING(1..15);
--   type EMP_NAME     is array(1..10) of EMP_NAME_CHARACTER;
--   type PRODUCT_NAME is array ( NATURAL range 0..20 ) of CHARACTER;

-- Routines to convert between program array types and the SQL_OBJECT internal
-- type are produced by instantiating generic functions with the program array
-- type.  The generic formal parameter for the program array type is declared
-- as "array (index_subtype) of component_type", where index_subtype and
-- component_type are also generic formal parameters.  (component_type is not
-- required for the special case of strings of CHARACTERs).

-- In order for the actual program array subtype used in the instantiation to
-- match the generic formal parameter, it is necessary that the actual array
-- have the same bounds as the index_subtype actual parameter.  If the actual
-- index subtype is anonymous, the user has not given us an appropriate
-- subtype to use for the index_subtype actual parameter.  So, we generate
-- one.  For the above examples, we would generate:
--
--   subtype DEPT_NAME_INDEX    is POSITIVE range 1 .. 15;
--   subtype EMP_NAME_INDEX     is INTEGER  range 1 .. 10;
--   subtype PRODUCT_NAME_INDEX is NATURAL  range 0 .. 20;
--
```

```
-- The type mark used in the subtype declaration is (in the same order as the
-- examples):
--
-- (1) For a derived array type, the same as the type mark that would be used
--     for the parent array type, unless the parent array type is declared by
--     an unconstrained array definition, in which case the type mark of its
--     index subtype definition is used (the index subtype of STRING is
--     POSITIVE)
--
-- (2) For an array type declared with an index range, INTEGER (the only form
--     of index range currently supported by the application scanner is low ..
--     high, where both low and high are integers)
--
-- (3) For an array type declared with an index subtype indication, the type
--     mark from the subtype indication

-- The following information is required to know how to generate these subtype
-- declarations:
--
-- (1) The name of the package in which the array type is declared - subtypes
--     are generated within nested packages corresponding to the packages in
--     which their arrays are declared, to avoid name conflicts caused by two
--     array types with the same name, but declared in different packages
--
-- (2) The name of the array type
--
-- (3) The type mark to use in the subtype declaration
--
-- (4) The bounds of the index

-- All this information is present in the ACCESS_TYPE_DESCRIPTOR for the array
-- type.  To indicate that an index subtype declaration must be generated,
-- INDEX_SUBTYPE.REQUIRED_FOR is called with the appropriate ACCESS_TYPE_-
-- DESCRIPTOR for the array type (ACCESS_STRING_DESCRIPTOR is a subtype of
-- ACCESS_TYPE_DESCRIPTOR that includes only descriptors of strings):

  procedure REQUIRED_FOR
            ( ARRAY_TYPE : DDL_DEFINITIONS.ACCESS_STRING_DESCRIPTOR );

-- INDEX_SUBTYPE.POST_PROCESSING is called to produce the index subtype
-- declarations in the generated package.

  procedure POST_PROCESSING;

end INDEX_SUBTYPE;
```

3.11.25 package indexb.ada

```
-- indexb.ada - post process data strucs for generated index subtypes needed
```

```
with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, DATABASE;
use  TEXT_PRINT;
package body INDEX_SUBTYPE is

   use DDL_DEFINITIONS;
   use DATABASE;

   type REQUIRED_FOR_ENTRY_RECORD;
   type REQUIRED_FOR_ENTRY is access REQUIRED_FOR_ENTRY_RECORD;

   type REQUIRED_FOR_ENTRY_RECORD is
      record
         ARRAY_TYPE         : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                                    DUMMY.ACCESS_TYPE_DESCRIPTOR;
         NEXT_REQUIRED_FOR     : REQUIRED_FOR_ENTRY;
      end record;

   REQUIRED_FOR_LIST : REQUIRED_FOR_ENTRY := new REQUIRED_FOR_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.SCHEMA_UNIT.NAME.all > RIGHT.SCHEMA_UNIT.NAME.all then
      return TRUE;
   elsif LEFT.SCHEMA_UNIT /= RIGHT.SCHEMA_UNIT then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";

procedure REQUIRED_FOR
   (ARRAY_TYPE : DDL_DEFINITIONS.ACCESS_STRING_DESCRIPTOR) is
   TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST;
   -- Order list by fully-qualified array type name.
begin
   while TRACER.NEXT_REQUIRED_FOR /= null and then
      ARRAY_TYPE.FULL_NAME >=
      TRACER.NEXT_REQUIRED_FOR.ARRAY_TYPE.FULL_NAME loop
      TRACER := TRACER.NEXT_REQUIRED_FOR;
   end loop;
   if ARRAY_TYPE /= TRACER.ARRAY_TYPE then
      TRACER.NEXT_REQUIRED_FOR :=
         new REQUIRED_FOR_ENTRY_RECORD'
         (ARRAY_TYPE         => ARRAY_TYPE,
          NEXT_REQUIRED_FOR => TRACER.NEXT_REQUIRED_FOR);
```

```
      end if;
   end REQUIRED_FOR;

   procedure POST_PROCESSING is
      TRACER         : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
      CURRENT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
   begin
      while TRACER /= null loop
         CURRENT_SCHEMA := TRACER.ARRAY_TYPE.FULL_NAME.SCHEMA_UNIT;
         SET_INDENT (4);
         PRINT ("package ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all) & "_INDEX_PACKAGE ");
         PRINT ("is");
         PRINT_LINE;
         while TRACER /= null and then
            TRACER.ARRAY_TYPE.FULL_NAME.SCHEMA_UNIT = CURRENT_SCHEMA loop
            SET_INDENT (6);
            PRINT ("subtype ");
            PRINT (STRING(TRACER.ARRAY_TYPE.FULL_NAME.NAME.all) & "_INDEX ");
            PRINT ("is ");
            PRINT (STRING(TRACER.ARRAY_TYPE.INDEX_TYPE.FULL_NAME.NAME.all));
            PRINT (" range ");
            PRINT (DATABASE.INT'IMAGE(TRACER.ARRAY_TYPE.ARRAY_RANGE_LO));
            PRINT (" .. ");
            PRINT (DATABASE.INT'IMAGE(TRACER.ARRAY_TYPE.ARRAY_RANGE_HI));
            PRINT (";");
            PRINT_LINE;
            TRACER := TRACER.NEXT_REQUIRED_FOR;
         end loop;
         SET_INDENT (4);
         PRINT ("end ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all) & "_INDEX_PACKAGE;");
         PRINT_LINE;
         BLANK_LINE;
      end loop;
   end POST_PROCESSING;

   end INDEX_SUBTYPE;
```

3.11.26 package dbtypes.ada

```
-- dbtypes.ada - post process data strucs for strongly typed database types

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package DATABASE_TYPE is

-- SQL operations can be performed between columns (e.g., MAX_TEMP and
-- NUMBER_LIVING_AT_HOME) and program variables (e.g., CURRENT_TEMP, of type
-- TEMPERATURE, and NUMBER_OF_CHILDREN, of type CHILDREN_COUNT) as in
```

```
--
--    ... WHERE => MAX_TEMP < CURRENT_TEMP ...
--    ... WHERE => NUMBER_LIVING_AT_HOME < NUMBER_OF_CHILDREN ...

-- The above examples imply that we have the following functions defined
-- ("some_type" to be discussed):
--
--    function "<" ( LEFT : some_type ; RIGHT : TEMPERATURE )    return ...
--    function "<" ( LEFT : some_type ; RIGHT : CHILDREN_COUNT ) return ...

-- If "some_type" is the same in both functions, then we have a problem
-- compiling an operation where a literal is used instead of a program
-- variable (assuming that both TEMPERATURE and CHILDREN_COUNT are integer
-- types):
--
--    ... WHERE => MAX_TEMP < 32 ...
--
-- The parameter and result type profile of the above "<" matches both
-- functions shown if "some_type" is the same type in both functions.  This is
-- such a common thing to program that we don't want to make the user qualify
-- all his literals, so we have to find a way to make the above operation
-- compile.

-- Obviously, the "some_type" must be different in each function.  We use the
-- term "strongly typed database type" to refer to the "some_type".  Each
-- strongly typed database type corresponds to a program type, e.g., the
-- MAX_TEMP column must have been defined of program type TEMPERATURE in order
-- for it to be comparable with the CURRENT_TEMP program variable, and the
-- MAX_TEMP function returns the strongly typed database type corresponding to
-- program type TEMPERATURE.

-- Objects of a strongly typed database type are actually data structures that
-- describe parts of an SQL statement, such as a column name (in these
-- examples), an operation on several operands, etc.  The actual details of
-- this data structure are not important to the application scanner; they are
-- embodied in the generics that are instantiated by the generated code.  (The
-- application scanner only has to know how to instantiate the generics, not
-- what's inside of them.)

-- All strongly typed database types are derived from type TYPED_SQL_OBJECT.
-- The use of derived types provides the convenience of deriving certain
-- handy conversion functions.  Again, these details are not important to the
-- application scanner; they are merely provided for background.

-- For each program type x that has a corresponding strongly typed database
-- type used as a parameter or return type of a generated subprogram, the
-- following statement is generated:
--
--    type x_TYPE is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;
```

```
-- These statements are separated into different packages nested within the
-- generated package, corresponding to the packages in which the program types
-- are declared.  This is done to avoid name conflicts where types with
-- identical names are declared in different source packages.

-- The information required to know how to generate the declarations of the
-- strongly typed database types therefore consists of:
--
-- (1) Name of the package in which the program type is declared
--
-- (2) Name of the program type

-- This information is included within the ACCESS_FULL_NAME_DESCRIPTOR for the
-- type.  To indicate that a program type requires a corresponding strongly
-- typed database type, DATABASE_TYPE.REQUIRED_FOR is called with the ACCESS_-
-- FULL_NAME_DESCRIPTOR for the program type:

  procedure REQUIRED_FOR
            ( PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- Three different post processing steps are required for the data structures
-- built to remember which strongly typed database types must be generated:
--
-- (1) The type declarations are produced
--
-- (2) In order to be able to take advantage of the handy conversion functions
--     derived by the type declarations, they must be directly visible.
--     Consequently, use clauses for the type declaration packages are
--     generated.  The package generated for program types declared in source
--     package p is named p_TYPE_PACKAGE, so the use clause produced is
--
--       use p_TYPE_PACKAGE, ...;
--
-- (3) The p_TYPE_PACKAGEs are actually produced inside of another package,
--     named ADA_SQL.  Step (2) produces a use clause within ADA_SQL, making
--     the handy conversion functions directly visible from the rest of that
--     package.  Direct visibility is also required outside of the ADA_SQL
--     package, so this step produces a use clause of the form
--
--       use ADA_SQL.p_TYPE_PACKAGE, ...;
--
-- Note: I think that the use clause produced by step (3) would also be
-- acceptable at the point where step (2) is used (I haven't tried compiling
-- it to make sure), but I have retained three steps for aesthetic and other
-- (equally frivolous) reasons.

-- The routines that cause the appropriate generated output to be produced for
-- each of these steps are, respectively:
```

```
  procedure POST_PROCESSING_TO_PRODUCE_TYPE_DECLARATIONS;

  procedure POST_PROCESSING_TO_PRODUCE_UNQUALIFIED_USE_CLAUSE;

  procedure POST_PROCESSING_TO_PRODUCE_QUALIFIED_USE_CLAUSE;

end DATABASE_TYPE;
```

3.11.27 package dbtypeb.ada

```
-- dbtypeb.ada - post process data strucs for strongly typed database types

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY;
use  TEXT_PRINT;
package body DATABASE_TYPE is

   use DDL_DEFINITIONS;

   type REQUIRED_FOR_ENTRY_RECORD;
   type REQUIRED_FOR_ENTRY is access REQUIRED_FOR_ENTRY_RECORD;

   type REQUIRED_FOR_ENTRY_RECORD is
      record
         FULL_NAME_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                   DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         NEXT_REQUIRED_FOR    : REQUIRED_FOR_ENTRY;
      end record;

   REQUIRED_FOR_LIST : REQUIRED_FOR_ENTRY := new REQUIRED_FOR_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.SCHEMA_UNIT.NAME.all > RIGHT.SCHEMA_UNIT.NAME.all then
      return TRUE;
   elsif LEFT.SCHEMA_UNIT /= RIGHT.SCHEMA_UNIT then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";

procedure REQUIRED_FOR
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST;
   -- Order list by fully-qualified program type name.
begin
```

```
      while TRACER.NEXT_REQUIRED_FOR /= null and then
         PROGRAM_TYPE >= TRACER.NEXT_REQUIRED_FOR.FULL_NAME_DESCRIPTOR loop
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
      if PROGRAM_TYPE /= TRACER.FULL_NAME_DESCRIPTOR then
         TRACER.NEXT_REQUIRED_FOR :=
            new REQUIRED_FOR_ENTRY_RECORD'
               (FULL_NAME_DESCRIPTOR => PROGRAM_TYPE,
                NEXT_REQUIRED_FOR    => TRACER.NEXT_REQUIRED_FOR);
      end if;
   end REQUIRED_FOR;

   procedure POST_PROCESSING_TO_PRODUCE_TYPE_DECLARATIONS is
      TRACER         : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
      CURRENT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
   begin
      while TRACER /= null loop
         CURRENT_SCHEMA := TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT;
         SET_INDENT (4);
         PRINT ("package ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all) & "_TYPE_PACKAGE ");
         PRINT ("is");
         PRINT_LINE;
         while TRACER /= null and then
            TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT = CURRENT_SCHEMA loop
            SET_INDENT (6);
            PRINT ("type ");
            PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all) & "_TYPE ");
            PRINT ("is new ADA_SQL_FUNCTIONS.TYPED_SQL_OBJECT;");
            PRINT_LINE;
            TRACER := TRACER.NEXT_REQUIRED_FOR;
         end loop;
         SET_INDENT (4);
         PRINT ("end ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all) & "_TYPE_PACKAGE;");
         PRINT_LINE;
         BLANK_LINE;
      end loop;
   end POST_PROCESSING_TO_PRODUCE_TYPE_DECLARATIONS;

   procedure POST_PROCESSING_TO_PRODUCE_UNQUALIFIED_USE_CLAUSE is
      TRACER         : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
      CURRENT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
   begin
      if TRACER /= null then
         SET_INDENT (4);
         PRINT ("use");
         while TRACER /= null loop
            CURRENT_SCHEMA := TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT;
```

```
         PRINT (" ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all) & "_TYPE_PACKAGE");
         while TRACER /= null and then
            TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT = CURRENT_SCHEMA loop
            TRACER := TRACER.NEXT_REQUIRED_FOR;
         end loop;
         if TRACER /= null then
            PRINT (",");
         end if;
      end loop;
      PRINT (";");
      PRINT_LINE;
      BLANK_LINE;
   end if;
end POST_PROCESSING_TO_PRODUCE_UNQUALIFIED_USE_CLAUSE;

procedure POST_PROCESSING_TO_PRODUCE_QUALIFIED_USE_CLAUSE is
   TRACER         : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
   CURRENT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
begin
   if TRACER /= null then
      SET_INDENT (2);
      PRINT ("use");
      while TRACER /= null loop
         CURRENT_SCHEMA := TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT;
         PRINT (" ADA_SQL." & STRING(CURRENT_SCHEMA.NAME.all)  &
                "_TYPE_PACKAGE");
         while TRACER /= null and then
            TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT = CURRENT_SCHEMA loop
            TRACER := TRACER.NEXT_REQUIRED_FOR;
         end loop;
         if TRACER /= null then
            PRINT (",");
         end if;
      end loop;
      PRINT (";");
      PRINT_LINE;
      BLANK_LINE;
   end if;
end POST_PROCESSING_TO_PRODUCE_QUALIFIED_USE_CLAUSE;

end DATABASE_TYPE;
```

3.11.28 package comptos.ada

```
-- comptos.ada - post process data strucs for CONVERT_COMPONENT_TO_CHARACTER

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package CONVERT_COMPONENT_TO_CHARACTER is
```

```
-- Ada/SQL permits strings to be arrays with components of any type derived
-- from CHARACTER.  In its internal data structures, Ada/SQL stores strings as
-- STRINGs.  An array program value is converted to its internal
-- representation by a function instantiated from a generic string conversion
-- function.  There is one string conversion function instantiated for each
-- program string type that must be converted to internal representation.

-- If the component type of the program string type is not CHARACTER, then the
-- string conversion function for that type must convert the program value
-- character by character, explicitly converting each program component to
-- type CHARACTER.  This explicit conversion is performed by a function called
-- CONVERT_COMPONENT_TO_CHARACTER, which is a generic formal subprogram to
-- the generic string conversion function.  The application scanner generates
-- the required subprograms named CONVERT_COMPONENT_TO_CHARACTER, so that each
-- string conversion function instantiation uses the correct component
-- conversion function by default (no actual parameter need be supplied to
-- the instantiation for the CONVERT_COMPONENT_TO_CHARACTER generic formal
-- subprogram.)

-- There is one CONVERT_COMPONENT_TO_CHARACTER function generated for each
-- type, other than CHARACTER, used as the component type of a string program
-- type that must be converted to internal representation.  Since the
-- functions rely on the fact that the component type is derived from
-- CHARACTER, they cannot be merely instantiated from generics, but must be
-- completely written.  In what follows, type_name represents the fully
-- qualified name of a component type.  If the type is defined in a DDL
-- package, type_name will be of the form library_unit.ADA_SQL.type_simple_-
-- name.  If the type is defined in a predefined package, type_name will be
-- of the form library_unit.type_simple_name.

-- The specification of each CONVERT_COMPONENT_TO_CHARACTER function is:
--
--    function CONVERT_COMPONENT_TO_CHARACTER ( C: type_name )
--     return CHARACTER;

-- The corresponding body is:
--
--    function CONVERT_COMPONENT_TO_CHARACTER ( C: type_name )
--     return CHARACTER is
--    begin
--      return CHARACTER ( C );
--    end CONVERT_COMPONENT_TO_CHARACTER;

-- The only information required to produce each CONVERT_COMPONENT_TO_-
-- CHARACTER function is the fully qualified name of the type involved.  This
-- information is found in the ACCESS_FULL_NAME_DESCRIPTOR for the type, and
-- it is a pointer to that data structure that is passed to CONVERT_-
-- COMPONENT_TO_CHARACTER.REQUIRED_FOR to indicate that a component conversion
-- function is to be generated for the indicated type.  CONVERT_COMPONENT_TO_-
```

```
-- CHARACTER.REQUIRED_FOR is called whenever it is determined that a component
-- conversion function is required; it automatically avoids generating
-- duplicate functions.

  procedure REQUIRED_FOR
           ( COMPONENT_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- There are two post processing steps for the CONVERT_COMPONENT_TO_CHARACTER
-- functions: producing the specifications and producing the bodies.  These
-- two steps are performed by CONVERT_COMPONENT_TO_CHARACTER.SPEC_POST_-
-- PROCESSING and CONVERT_COMPONENT_TO_CHARACTER.BODY_POST_PROCESSING.

  procedure SPEC_POST_PROCESSING;

  procedure BODY_POST_PROCESSING;

end CONVERT_COMPONENT_TO_CHARACTER;
```

3.11.29 package comptob.ada

```
-- comptob.ada - post process data strucs for CONVERT_COMPONENT_TO_CHARACTER

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY;
use  TEXT_PRINT;
package body CONVERT_COMPONENT_TO_CHARACTER is

   use DDL_DEFINITIONS;

   type REQUIRED_FOR_ENTRY_RECORD;
   type REQUIRED_FOR_ENTRY is access REQUIRED_FOR_ENTRY_RECORD;

   type REQUIRED_FOR_ENTRY_RECORD is
      record
         FULL_NAME_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                      DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         NEXT_REQUIRED_FOR    : REQUIRED_FOR_ENTRY;
      end record;

   REQUIRED_FOR_LIST : REQUIRED_FOR_ENTRY := new REQUIRED_FOR_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.FULL_PACKAGE_NAME.all > RIGHT.FULL_PACKAGE_NAME.all then
      return TRUE;
   elsif LEFT.FULL_PACKAGE_NAME.all /= RIGHT.FULL_PACKAGE_NAME.all then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
```

```
      else
         return FALSE;
      end if;
   end ">=";

   procedure REQUIRED_FOR
      (COMPONENT_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST;
      -- Order list by fully-qualified component type name.
   begin
      while TRACER.NEXT_REQUIRED_FOR /= null and then
         COMPONENT_TYPE >= TRACER.NEXT_REQUIRED_FOR.FULL_NAME_DESCRIPTOR loop
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
      if COMPONENT_TYPE /= TRACER.FULL_NAME_DESCRIPTOR then
         TRACER.NEXT_REQUIRED_FOR :=
            new REQUIRED_FOR_ENTRY_RECORD'
               (FULL_NAME_DESCRIPTOR => COMPONENT_TYPE,
                NEXT_REQUIRED_FOR    => TRACER.NEXT_REQUIRED_FOR);
      end if;
   end REQUIRED_FOR;

   procedure SPEC_POST_PROCESSING is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
   begin
      while TRACER /= null loop
         SET_INDENT (2);
         PRINT ("function CONVERT_COMPONENT_TO_CHARACTER");
         PRINT_LINE;
         SET_INDENT (4);
         PRINT ("( C : ");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
         PRINT (" )");
         PRINT_LINE;
         PRINT ("return CHARACTER;");
         PRINT_LINE;
         BLANK_LINE;
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
   end SPEC_POST_PROCESSING;

   procedure BODY_POST_PROCESSING is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
   begin
      while TRACER /= null loop
         SET_INDENT (2);
         PRINT ("function CONVERT_COMPONENT_TO_CHARACTER");
         PRINT_LINE;
```

```
      SET_INDENT (4);
      PRINT ("( C : ");
      PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
      PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
      PRINT (" )");
      PRINT_LINE;
      PRINT ("return CHARACTER is");
      PRINT_LINE;
      SET_INDENT (2);
      PRINT ("begin");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("return CHARACTER ( C );");
      PRINT_LINE;
      SET_INDENT (2);
      PRINT ("end CONVERT_COMPONENT_TO_CHARACTER;");
      PRINT_LINE;
      BLANK_LINE;
      TRACER := TRACER.NEXT_REQUIRED_FOR;
   end loop;
end BODY_POST_PROCESSING;

end CONVERT_COMPONENT_TO_CHARACTER;
```

3.11.30 package chartos.ada

```
-- chartos.ada - post process data strucs for CONVERT_CHARACTER_TO_COMPONENT

with DDL_DEFINITIONS;
package CONVERT_CHARACTER_TO_COMPONENT is

-- Ada/SQL permits strings to be arrays with components of any type derived
-- from CHARACTER.  When processing data returned from the database, Ada/SQL
-- stores strings as STRINGs.  For passing it back to an application program,
-- this returned data is converted to its program array type by an INTO
-- procedure instantiated from a generic string INTO procedure.  There is one
-- string INTO procedure instantiated for each program string type that may be
-- returned to the application program.

-- The generic INTO procedure converts the returned database STRING into the
-- program array type character by character, explicitly converting each
-- program component to type CHARACTER.  (This conversion is unnecessary for
-- program array types of CHARACTER, but I figured that the INTO procedure
-- would probably have to be looking at each character of the result anyway,
-- in order to decode where a particular column result stops and the next one
-- starts, so why not let it call the conversion routine in all instances?  If
-- the conversion routine is INLINEd, then it doesn't generate any code
-- anyway.  I did not bother with pragma INLINE in the example, but it could
-- be easily added since the entire generated package is now [will soon be]
-- magically produced by computer.)
```

```
-- This explicit conversion is performed by a function called CONVERT_-
-- CHARACTER_TO_COMPONENT, which is a generic formal subprogram to the generic
-- INTO procedure.  The application scanner generates the required functions
-- named CONVERT_CHARACTER_TO_COMPONENT, so that each INTO procedure
-- instantiation uses the correct component conversion function by default (no
-- actual parameter need by supplied to the instantiation for the CONVERT_-
-- CHARACTER_TO_COMPONENT generic formal subprogram.)

-- There is one CONVERT_CHARACTER_TO_COMPONENT function generated for each
-- type, including CHARACTER, used as the component type of a string program
-- type that is retrieved from the database.  Since the functions rely on the
-- fact that the component type is derived from CHARACTER, they cannot be
-- merely instantiated from generics, but must be completely written.  In
-- what follows, type_name represents the fully qualified name of a component
-- type.  If the type is defined in a DDL package, type_name will be of the
-- form library_unit.ADA_SQL.type_simple_name.  If the type is defined in a
-- predefined package, type_name will be of the form library_unit.type_-
-- simple_name.  This includes STANDARD.CHARACTER -- the hand-generated
-- package for the runtime example used a type_name of CHARACTER, but
-- STANDARD.CHARACTER is easier to program (no need to check for special
-- case), and may be used.

-- The specification of each CONVERT_CHARACTER_TO_COMPONENT function is:
--
--   function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
--    return type_name;

-- The corresponding body is:
--
--   function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
--    return type_name is
--   begin
--     return type_name ( C );
--   end CONVERT_CHARACTER_TO_COMPONENT;

-- Where type_name was CHARACTER, the hand-generated package for the runtime
-- example did not apply the conversion function in the body, saying just
-- "return C;".  There is certainly no harm in applying a type conversion
-- function to STANDARD.CHARACTER, and this may be done, rather than program
-- for the special case.

-- The only information required to produce each CONVERT_CHARACTER_TO_-
-- COMPONENT function is the fully qualified name of the type involved.  This
-- information is found in the ACCESS_FULL_NAME_DESCRIPTOR for the type, and
-- it is a pointer to that data structure that is passed to CONVERT_-
-- CHARACTER_TO_COMPONENT.REQUIRED_FOR to indicate that a component conversion
-- function is to be generated for the indicated type.  CONVERT_CHARACTER_TO_-
-- COMPONENT.REQUIRED_FOR is called whenever it is determined that a component
-- conversion function is required; it automatically avoids generating
```

```
-- duplicate functions.

  procedure REQUIRED_FOR
            ( COMPONENT_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- There are two post processing steps for the CONVERT_CHARACTER_TO_COMPONENT
-- functions: producing the specifications and producing the bodies.  These
-- two steps are performed by CONVERT_CHARACTER_TO_COMPONENT.SPEC_POST_-
-- PROCESSING and CONVERT_CHARACTER_TO_COMPONENT.BODY_POST_PROCESSING.

  procedure SPEC_POST_PROCESSING;

  procedure BODY_POST_PROCESSING;

end CONVERT_CHARACTER_TO_COMPONENT;
```

3.11.31 package chartob.ada

```
-- chartob.ada - post process data strucs for CONVERT_CHARACTER_TO_COMPONENT

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY;
use  TEXT_PRINT;
package body CONVERT_CHARACTER_TO_COMPONENT is

   use DDL_DEFINITIONS;

   type REQUIRED_FOR_ENTRY_RECORD;
   type REQUIRED_FOR_ENTRY is access REQUIRED_FOR_ENTRY_RECORD;

   type REQUIRED_FOR_ENTRY_RECORD is
      record
         FULL_NAME_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                   DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         NEXT_REQUIRED_FOR    : REQUIRED_FOR_ENTRY;
      end record;

   REQUIRED_FOR_LIST : REQUIRED_FOR_ENTRY := new REQUIRED_FOR_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.FULL_PACKAGE_NAME.all > RIGHT.FULL_PACKAGE_NAME.all then
      return TRUE;
   elsif LEFT.FULL_PACKAGE_NAME.all /= RIGHT.FULL_PACKAGE_NAME.all then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
```

```
      end if;
   end ">=";

   procedure REQUIRED_FOR
      (COMPONENT_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST;
      -- Order list by fully-qualified component type name.
   begin
      while TRACER.NEXT_REQUIRED_FOR /= null and then
         COMPONENT_TYPE >= TRACER.NEXT_REQUIRED_FOR.FULL_NAME_DESCRIPTOR loop
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
      if COMPONENT_TYPE /= TRACER.FULL_NAME_DESCRIPTOR then
         TRACER.NEXT_REQUIRED_FOR :=
            new REQUIRED_FOR_ENTRY_RECORD'
               (FULL_NAME_DESCRIPTOR => COMPONENT_TYPE,
                NEXT_REQUIRED_FOR    => TRACER.NEXT_REQUIRED_FOR);
      end if;
   end REQUIRED_FOR;

   procedure SPEC_POST_PROCESSING is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
   begin
      while TRACER /= null loop
         SET_INDENT (2);
         PRINT ("function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )");
         PRINT_LINE;
         PRINT ("return ");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
         PRINT (";");
         PRINT_LINE;
         BLANK_LINE;
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
   end SPEC_POST_PROCESSING;

   procedure BODY_POST_PROCESSING is
      TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
   begin
      while TRACER /= null loop
         SET_INDENT (2);
         PRINT ("function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )");
         PRINT_LINE;
         PRINT ("return ");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
         PRINT (" is");
         PRINT_LINE;
```

```
      PRINT ("begin");
      PRINT_LINE;
      PRINT ("return ");
      PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
      PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
      PRINT (" ( C );");
      PRINT_LINE;
      PRINT ("end CONVERT_CHARACTER_TO_COMPONENT;");
      PRINT_LINE;
      BLANK_LINE;
      TRACER := TRACER.NEXT_REQUIRED_FOR;
   end loop;
end BODY_POST_PROCESSING;

end CONVERT_CHARACTER_TO_COMPONENT;
```

3.11.32 package tables.ada

```
-- tables.ada - miscellaneous routines for handling table names

with DDL_DEFINITIONS;
package TABLE is

-- The DDL reader requires that table names be unique within authorization
-- identifier.  In this implementation, however, the application scanner does
-- not recognize authorization identifiers as part of table names.  It is
-- therefore possible for references to tables to be ambiguous.  We do not
-- allow this.  When processing a table name, there are therefore three
-- possible outcomes, of which only the last is not an error, as given by
-- values of the following enumeration type:

  type NAME_STATUS is ( NAME_UNDEFINED , NAME_AMBIGUOUS , NAME_UNIQUE );

-- TABLE.DESCRIPTOR_FOR determines the TABLE.NAME_STATUS for the given table
-- name (specified in its string representation), and locates the ACCESS_-
-- TYPE_DESCRIPTOR for the table (value valid if and only if TABLE.NAME_-
-- UNIQUE).

  procedure DESCRIPTOR_FOR
            ( NAME       : in  STRING;
              STATUS     : out NAME_STATUS;
              DESCRIPTOR : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR );

end TABLE;
```

3.11.33 package tableb.ada

```
with DDL_DEFINITIONS, DDL_VARIABLES;
use  DDL_DEFINITIONS, DDL_VARIABLES;
```

```
package body TABLE is
  procedure DESCRIPTOR_FOR
           (NAME            : in     STRING;
            STATUS          :    out NAME_STATUS;
            DESCRIPTOR      :    out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR) is

    TABLE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                DDL_VARIABLES.FIRST_TABLE;
    COUNT     : NATURAL := 0;

  begin
    while TABLE_DES /= null loop
      if NAME = STRING (TABLE_DES.FULL_NAME.NAME.all) then
        DESCRIPTOR := TABLE_DES;
        COUNT := COUNT + 1;
      end if;
      TABLE_DES := TABLE_DES.NEXT_TYPE;
    end loop;
    if COUNT = 0 then
      DESCRIPTOR := null;
      STATUS := NAME_UNDEFINED;
    elsif COUNT = 1 then
      STATUS := NAME_UNIQUE;
    else
      DESCRIPTOR := null;
      STATUS := NAME_AMBIGUOUS;
    end if;
  end DESCRIPTOR_FOR;
end TABLE;
```

3.11.34 package pdtypes.ada

```
-- pdtypes.ada - functions to identify predefined (STANDARD or DATABASE) types

with DDL_DEFINITIONS;
package PREDEFINED_TYPE is

-- This package provides access to the ACCESS_TYPE_DESCRIPTORs of certain
-- predefined (e.g., in the packages STANDARD and DATABASE) types.  Since we
-- use ACCESS_TYPE_DESCRIPTOR values in comparisons, these values must be the
-- actual unique descriptors created to represent these types in the DDL
-- data structures.

   package STANDARD is

      function INTEGER          return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function FLOAT            return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function STRING           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function CHARACTER        return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function BOOLEAN          return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
```

```
   end STANDARD;

   package DATABASE is

      function INT                return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function DOUBLE_PRECISION   return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function CHAR               return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      function COLUMN_NUMBER      return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

   end DATABASE;

   package CURSOR_DEFINITION is

      function CURSOR_NAME        return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

   end CURSOR_DEFINITION;

end PREDEFINED_TYPE;
```

3.11.35 package pdtypeb.ada

```
-- pdtypes.ada - functions to identify predefined (STANDARD or DATABASE) types

with DDL_DEFINITIONS, DDL_VARIABLES;
use  DDL_DEFINITIONS;

package body PREDEFINED_TYPE is

  STANDARD_INTEGER           : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  STANDARD_FLOAT             : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  STANDARD_STRING            : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  STANDARD_CHARACTER         : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  STANDARD_BOOLEAN           : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  DATABASE_INT               : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  DATABASE_DOUBLE_PRECISION  : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  DATABASE_CHAR              : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  DATABASE_COLUMN_NUMBER     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
  CDEF_CURSOR_NAME           : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;

  function FIND_TYPE_DESCRIPTOR
          (PAK_NAME  : STRING;
           TYPE_NAME : STRING)
           return      DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is

    TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
               DDL_VARIABLES.FIRST_TYPE;

  begin
    while TYPE_DES /= null loop
      if PAK_NAME = STRING (TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all) and then
```

```
        TYPE_NAME = STRING (TYPE_DES.FULL_NAME.NAME.all) then
       return TYPE_DES;
     end if;
     TYPE_DES := TYPE_DES.NEXT_TYPE;
   end loop;
   return null;
end FIND_TYPE_DESCRIPTOR;

package body STANDARD is

  function INTEGER
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if STANDARD_INTEGER = null then
      STANDARD_INTEGER := FIND_TYPE_DESCRIPTOR ("STANDARD", "INTEGER");
    end if;
    return STANDARD_INTEGER;
  end INTEGER;

  function FLOAT
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if STANDARD_FLOAT = null then
      STANDARD_FLOAT := FIND_TYPE_DESCRIPTOR ("STANDARD", "FLOAT");
    end if;
    return STANDARD_FLOAT;
  end FLOAT;

  function STRING
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if STANDARD_STRING = null then
      STANDARD_STRING := FIND_TYPE_DESCRIPTOR ("STANDARD", "STRING");
    end if;
    return STANDARD_STRING;
  end STRING;

  function CHARACTER
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if STANDARD_CHARACTER = null then
      STANDARD_CHARACTER := FIND_TYPE_DESCRIPTOR ("STANDARD", "CHARACTER");
    end if;
    return STANDARD_CHARACTER;
  end CHARACTER;

  function BOOLEAN
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
```

```
    if STANDARD_BOOLEAN = null then
      STANDARD_BOOLEAN := FIND_TYPE_DESCRIPTOR ("STANDARD", "BOOLEAN");
    end if;
    return STANDARD_BOOLEAN;
  end BOOLEAN;

end STANDARD;

package body DATABASE is

  function INT
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if DATABASE_INT = null then
      DATABASE_INT := FIND_TYPE_DESCRIPTOR ("DATABASE", "INT");
    end if;
    return DATABASE_INT;
  end INT;

  function DOUBLE_PRECISION
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if DATABASE_DOUBLE_PRECISION = null then
      DATABASE_DOUBLE_PRECISION :=
                   FIND_TYPE_DESCRIPTOR ("DATABASE", "DOUBLE_PRECISION");
    end if;
    return DATABASE_DOUBLE_PRECISION;
  end DOUBLE_PRECISION;

  function CHAR
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if DATABASE_CHAR = null then
      DATABASE_CHAR := FIND_TYPE_DESCRIPTOR ("DATABASE", "CHAR");
    end if;
    return DATABASE_CHAR;
  end CHAR;

  function COLUMN_NUMBER
           return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  begin
    if DATABASE_COLUMN_NUMBER = null then
      DATABASE_COLUMN_NUMBER :=
                  FIND_TYPE_DESCRIPTOR ("DATABASE", "COLUMN_NUMBER");
    end if;
    return DATABASE_COLUMN_NUMBER;
  end COLUMN_NUMBER;

end DATABASE;
```

```
  package body CURSOR_DEFINITION IS

    function CURSOR_NAME
             return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
    begin
      if CDEF_CURSOR_NAME = null then
        CDEF_CURSOR_NAME :=
              FIND_TYPE_DESCRIPTOR ("CURSOR_DEFINITION", "CURSOR_NAME");
      end if;
      return CDEF_CURSOR_NAME;
    end CURSOR_NAME;
  end CURSOR_DEFINITION;
end PREDEFINED_TYPE;
```

3.11.36 package ddl_add_des_spec.ada

```
with DDL_DEFINITIONS, DDL_VARIABLES;
use  DDL_DEFINITIONS, DDL_VARIABLES;

package ADD_DESCRIPTOR_ROUTINES is

  procedure ADD_YET_TO_DO_DESCRIPTOR
           (NEW_YET_TO_DO_DESCRIPTOR          : in out
                ACCESS_YET_TO_DO_DESCRIPTOR);

  procedure ADD_SCHEMA_UNIT_DESCRIPTOR
           (NEW_SCHEMA_UNIT_DESCRIPTOR        : in out
                ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure ADD_WITHED_UNIT_DESCRIPTOR
           (NEW_WITHED_UNIT_DESCRIPTOR        : in out
                ACCESS_WITHED_UNIT_DESCRIPTOR;
            OUR_SCHEMA_UNIT                   : in out
                ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure ADD_USED_PACKAGE_DESCRIPTOR
           (NEW_USED_PACKAGE_DESCRIPTOR       : in out
                ACCESS_USED_PACKAGE_DESCRIPTOR;
            OUR_SCHEMA_UNIT                   : in out
                ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure ADD_DECLARED_PACKAGE_DESCRIPTOR
           (NEW_DECLARED_PACKAGE_DESCRIPTOR   : in out
                ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
            OUR_SCHEMA_UNIT                   : in out
                ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure ADD_IDENTIFIER_DESCRIPTOR
           (NEW_IDENTIFIER_DESCRIPTOR         : in out
                ACCESS_IDENTIFIER_DESCRIPTOR);
```

```
  procedure ADD_FULL_NAME_DESCRIPTOR
           (NEW_FULL_NAME_DESCRIPTOR          : in out
                ACCESS_FULL_NAME_DESCRIPTOR;
            OUR_IDENTIFIER_DESCRIPTOR         : in out
                ACCESS_IDENTIFIER_DESCRIPTOR);

  procedure ADD_TYPE_DESCRIPTOR
           (NEW_TYPE_DESCRIPTOR               : in out
                ACCESS_TYPE_DESCRIPTOR);

  procedure ADD_VARIABLE_TYPE_DESCRIPTOR
           (NEW_TYPE_DESCRIPTOR               : in out
                ACCESS_TYPE_DESCRIPTOR);

  procedure ADD_RECORD_TYPE_DESCRIPTOR
           (NEW_TYPE_DESCRIPTOR               : in out
                ACCESS_TYPE_DESCRIPTOR);

  procedure ADD_LITERAL_DESCRIPTOR
           (NEW_LITERAL_DESCRIPTOR            : in out
                ACCESS_LITERAL_DESCRIPTOR;
            OUR_ENUMERATION_DES               : in out
                ACCESS_ENUMERATION_DESCRIPTOR);

  procedure ADD_ENUM_LIT_DESCRIPTOR
       (NEW_ENUM_LIT_DESCRIPTOR  : in out ACCESS_ENUM_LIT_DESCRIPTOR);

  procedure ADD_FULL_ENUM_LIT_DESCRIPTOR
       (NEW_FULL_ENUM_LIT_DESCRIPTOR : in out ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
        OUR_ENUM_LIT_DESCRIPTOR      : in out ACCESS_ENUM_LIT_DESCRIPTOR);

end ADD_DESCRIPTOR_ROUTINES;
```

3.11.37 package ddl_add_des.ada

```
package body ADD_DESCRIPTOR_ROUTINES is

----------------------------------------------------------------------------
--
-- ADD-YET_TO_DO_DESCRIPTOR
--
-- if this is the first yet-to-do defined set the first pointer
-- otherwise set the "next" pointer in the previously last yet-to-do to
--            point to this new yet-to-do
-- set the previous pointer in this new yet-to-do to point to the
--            old last yet-to-do
-- and now the new yet-to-do is the last one

  procedure ADD_YET_TO_DO_DESCRIPTOR
           (NEW_YET_TO_DO_DESCRIPTOR : in out ACCESS_YET_TO_DO_DESCRIPTOR) is
```

```
  begin
    if LAST_YET_TO_DO = null then
      FIRST_YET_TO_DO := NEW_YET_TO_DO_DESCRIPTOR;
    else
      LAST_YET_TO_DO.NEXT_YET_TO_DO := NEW_YET_TO_DO_DESCRIPTOR;
    end if;
    NEW_YET_TO_DO_DESCRIPTOR.PREVIOUS_YET_TO_DO := LAST_YET_TO_DO;
    LAST_YET_TO_DO := NEW_YET_TO_DO_DESCRIPTOR;
  end ADD_YET_TO_DO_DESCRIPTOR;

------------------------------------------------------------------------------
--
-- ADD_SCHEMA_UNIT_DESCRIPTOR
--
-- if this is the first schema unit defined set the first pointer
-- otherwise set the "next" pointer in the previously last schema unit to
--           point to this new schema unit
-- set the previous pointer in this new schema unit to point to the
--           old last schema unit
-- and now the new schema unit is the last one

  procedure ADD_SCHEMA_UNIT_DESCRIPTOR
        (NEW_SCHEMA_UNIT_DESCRIPTOR : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
  begin
    if LAST_SCHEMA_UNIT = null then
     FIRST_SCHEMA_UNIT := NEW_SCHEMA_UNIT_DESCRIPTOR;
    else
      LAST_SCHEMA_UNIT.NEXT_SCHEMA_UNIT := NEW_SCHEMA_UNIT_DESCRIPTOR;
    end if;
    NEW_SCHEMA_UNIT_DESCRIPTOR.PREVIOUS_SCHEMA_UNIT := LAST_SCHEMA_UNIT;
    LAST_SCHEMA_UNIT := NEW_SCHEMA_UNIT_DESCRIPTOR;
  end ADD_SCHEMA_UNIT_DESCRIPTOR;

------------------------------------------------------------------------------
--
-- ADD_WITHED_UNIT_DESCRIPTOR
--
-- if this is the first withed unit defined for this schema unit set the

--           first pointer
-- otherwise set the "next" pointer in the previously last withed unit to
--           point to this new withed unit
-- set the previous pointer in this new withed unit to point to the
--           old last withed unit
-- and now the new withed unit is the last one pointed to by the schema

  procedure ADD_WITHED_UNIT_DESCRIPTOR
      (NEW_WITHED_UNIT_DESCRIPTOR : in out ACCESS_WITHED_UNIT_DESCRIPTOR;
       OUR_SCHEMA_UNIT            : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
```

```
  begin
    if OUR_SCHEMA_UNIT.LAST_WITHED = null then
      OUR_SCHEMA_UNIT.FIRST_WITHED := NEW_WITHED_UNIT_DESCRIPTOR;
    else
      OUR_SCHEMA_UNIT.LAST_WITHED.NEXT_WITHED := NEW_WITHED_UNIT_DESCRIPTOR;
    end if;
    NEW_WITHED_UNIT_DESCRIPTOR.PREVIOUS_WITHED := OUR_SCHEMA_UNIT.LAST_WITHED;
    OUR_SCHEMA_UNIT.LAST_WITHED := NEW_WITHED_UNIT_DESCRIPTOR;
  end ADD_WITHED_UNIT_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_USED_PACKAGE_DESCRIPTOR
--
-- if this is the first used unit defined for this schema unit set the
--           first pointer
-- otherwise set the "next" pointer in the previously last used unit to
--           point to this new used unit
-- set the previous pointer in this new used unit to point to the
--           old last used unit
-- and now the new used unit is the last one pointed to by the schema

  procedure ADD_USED_PACKAGE_DESCRIPTOR
      (NEW_USED_PACKAGE_DESCRIPTOR   : in out ACCESS_USED_PACKAGE_DESCRIPTOR;
       OUR_SCHEMA_UNIT               : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
  begin
    if OUR_SCHEMA_UNIT.LAST_USED = null then
      OUR_SCHEMA_UNIT.FIRST_USED := NEW_USED_PACKAGE_DESCRIPTOR;
    else
      OUR_SCHEMA_UNIT.LAST_USED.NEXT_USED := NEW_USED_PACKAGE_DESCRIPTOR;
    end if;
    NEW_USED_PACKAGE_DESCRIPTOR.PREVIOUS_USED := OUR_SCHEMA_UNIT.LAST_USED;
    OUR_SCHEMA_UNIT.LAST_USED := NEW_USED_PACKAGE_DESCRIPTOR;
  end ADD_USED_PACKAGE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_DECLARED_PACKAGE_DESCRIPTOR
--

-- if this is the first declared package for this schema unit set the
--           first pointer
-- otherwise set the "next" pointer in the previously last declared package
--           to point to this new declared package
-- set the previous pointer in this new declared package to point to the
--           old last declared package
-- and now the new declared package is the last one pointed to by the schema

  procedure ADD_DECLARED_PACKAGE_DESCRIPTOR
```

```
    (NEW_DECLARED_PACKAGE_DESCRIPTOR  : in out
                                        ACCESS_DECLARED_PACKAGE_DESCRIPTOR;
     OUR_SCHEMA_UNIT : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
begin
  if OUR_SCHEMA_UNIT.LAST_DECLARED_PACKAGE = null then
    OUR_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE :=
                                        NEW_DECLARED_PACKAGE_DESCRIPTOR;
  else
    OUR_SCHEMA_UNIT.LAST_DECLARED_PACKAGE.NEXT_DECLARED :=
                                       NEW_DECLARED_PACKAGE_DESCRIPTOR;
  end if;
  NEW_DECLARED_PACKAGE_DESCRIPTOR.PREVIOUS_DECLARED :=
                                   OUR_SCHEMA_UNIT.LAST_DECLARED_PACKAGE;
  OUR_SCHEMA_UNIT.LAST_DECLARED_PACKAGE := NEW_DECLARED_PACKAGE_DESCRIPTOR;
end ADD_DECLARED_PACKAGE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_IDENTIFIER_DESCRIPTOR
--
-- if this is the first declared identifier set the first pointer
-- otherwise set the "next" pointer in the previously last identifier
--          to point to this new identifier
-- set the previous pointer in this new identifier to point to the
--          old last identifier
-- and now the new identifier is the last one

  procedure ADD_IDENTIFIER_DESCRIPTOR
      (NEW_IDENTIFIER_DESCRIPTOR  : in out ACCESS_IDENTIFIER_DESCRIPTOR) is
  begin
    if LAST_IDENTIFIER = null then
      FIRST_IDENTIFIER := NEW_IDENTIFIER_DESCRIPTOR;
    else
      LAST_IDENTIFIER.NEXT_IDENT := NEW_IDENTIFIER_DESCRIPTOR;
    end if;
    NEW_IDENTIFIER_DESCRIPTOR.PREVIOUS_IDENT := LAST_IDENTIFIER;
    LAST_IDENTIFIER := NEW_IDENTIFIER_DESCRIPTOR;
  end ADD_IDENTIFIER_DESCRIPTOR;


----------------------------------------------------------------------------
--
-- ADD_FULL_NAME_DESCRIPTOR
--
-- if this is the first declared full name for this identifier set the first
--             pointer
-- otherwise set the "next" pointer in the previously last full name
--           to point to this new full name
-- set the previous pointer in this new full name to point to the old last full
```

```
--             name in the identifier descriptor
-- and now the new full name is the last one for this identifier

  procedure ADD_FULL_NAME_DESCRIPTOR
      (NEW_FULL_NAME_DESCRIPTOR  : in out ACCESS_FULL_NAME_DESCRIPTOR;
       OUR_IDENTIFIER_DESCRIPTOR : in out ACCESS_IDENTIFIER_DESCRIPTOR) is
  begin
    if OUR_IDENTIFIER_DESCRIPTOR.LAST_FULL_NAME = null then
      OUR_IDENTIFIER_DESCRIPTOR.FIRST_FULL_NAME := NEW_FULL_NAME_DESCRIPTOR;
    else
      OUR_IDENTIFIER_DESCRIPTOR.LAST_FULL_NAME.NEXT_NAME :=
                                               NEW_FULL_NAME_DESCRIPTOR;
    end if;
    NEW_FULL_NAME_DESCRIPTOR.PREVIOUS_NAME :=
                                OUR_IDENTIFIER_DESCRIPTOR.LAST_FULL_NAME;
    OUR_IDENTIFIER_DESCRIPTOR.LAST_FULL_NAME := NEW_FULL_NAME_DESCRIPTOR;
  end ADD_FULL_NAME_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_TYPE_DESCRIPTOR
--
-- if this is the first type set the first pointer
-- otherwise set the "next" pointer in the previously last type to point
--      to this new type
-- set the previous pointer in this new type to point to the old last type
-- and now the new type is the last one

  procedure ADD_TYPE_DESCRIPTOR
      (NEW_TYPE_DESCRIPTOR  : in out ACCESS_TYPE_DESCRIPTOR) is
  begin
    if LAST_TYPE = null then
      FIRST_TYPE := NEW_TYPE_DESCRIPTOR;
    else
      LAST_TYPE.NEXT_TYPE := NEW_TYPE_DESCRIPTOR;
    end if;
    NEW_TYPE_DESCRIPTOR.PREVIOUS_TYPE := LAST_TYPE;
    LAST_TYPE := NEW_TYPE_DESCRIPTOR;
  end ADD_TYPE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_VARIABLE_TYPE_DESCRIPTOR
--
-- if this is the first variable set the first pointer
-- otherwise set the "next" pointer in the previously last variable to point
--      to this new variable
-- set the previous pointer in this new variable to point to the
--      old last variable
```

```
-- and now the new variable is the last one

  procedure ADD_VARIABLE_TYPE_DESCRIPTOR
      (NEW_TYPE_DESCRIPTOR  : in out ACCESS_TYPE_DESCRIPTOR) is
  begin
    if LAST_VARIABLE = null then
      FIRST_VARIABLE := NEW_TYPE_DESCRIPTOR;
    else
      LAST_VARIABLE.NEXT_TYPE := NEW_TYPE_DESCRIPTOR;
    end if;
    NEW_TYPE_DESCRIPTOR.PREVIOUS_TYPE := LAST_VARIABLE;
    LAST_VARIABLE := NEW_TYPE_DESCRIPTOR;
  end ADD_VARIABLE_TYPE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_RECORD_TYPE_DESCRIPTOR
--
-- if this is the first table set the first pointer
-- otherwise set the "next" pointer in the previously last table to point
--       to this new table
-- set the previous pointer in this new table to point to the old last table
-- and now the new table is the last one

  procedure ADD_RECORD_TYPE_DESCRIPTOR
      (NEW_TYPE_DESCRIPTOR  : in out ACCESS_TYPE_DESCRIPTOR) is
  begin
    if LAST_TABLE = null then
      FIRST_TABLE := NEW_TYPE_DESCRIPTOR;
    else
      LAST_TABLE.NEXT_TYPE := NEW_TYPE_DESCRIPTOR;
    end if;
    NEW_TYPE_DESCRIPTOR.PREVIOUS_TYPE := LAST_TABLE;
    LAST_TABLE := NEW_TYPE_DESCRIPTOR;
  end ADD_RECORD_TYPE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_LITERAL_DESCRIPTOR
--

-- if this is the first literal defined for this enumeration type set the
--           first pointer
-- otherwise set the "next" pointer in the previously last literal to
--           point to this new literal
-- set the previous pointer in this new literal to point to the
--           old last literal
-- and now the new literal is the last one pointed to by the enumeration type
```

```
  procedure ADD_LITERAL_DESCRIPTOR
      (NEW_LITERAL_DESCRIPTOR    : in out ACCESS_LITERAL_DESCRIPTOR;
       OUR_ENUMERATION_DES       : in out ACCESS_ENUMERATION_DESCRIPTOR) is
  begin
    if OUR_ENUMERATION_DES.LAST_LITERAL = null then
      OUR_ENUMERATION_DES.FIRST_LITERAL := NEW_LITERAL_DESCRIPTOR;
    else
      OUR_ENUMERATION_DES.LAST_LITERAL.NEXT_LITERAL := NEW_LITERAL_DESCRIPTOR;
    end if;
    NEW_LITERAL_DESCRIPTOR.PREVIOUS_LITERAL :=
                                          OUR_ENUMERATION_DES.LAST_LITERAL;
    OUR_ENUMERATION_DES.LAST_LITERAL := NEW_LITERAL_DESCRIPTOR;
  end ADD_LITERAL_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_ENUM_IDENT_DESCRIPTOR
--
-- if this is the first enumeration literal set the first pointer
-- otherwise set the "next" pointer in the previously last enumeration literal
--          to point to this new enumeration literal
-- set the previous pointer in this new enumeration literal to point to the
--          old last enumeration literal
-- and now the new enumeration literal is the last one

  procedure ADD_ENUM_LIT_DESCRIPTOR
      (NEW_ENUM_LIT_DESCRIPTOR   : in out ACCESS_ENUM_LIT_DESCRIPTOR) is
  begin
    if LAST_ENUM_LIT = null then
      FIRST_ENUM_LIT := NEW_ENUM_LIT_DESCRIPTOR;
    else
      LAST_ENUM_LIT.NEXT_ENUM_LIT := NEW_ENUM_LIT_DESCRIPTOR;
    end if;
    NEW_ENUM_LIT_DESCRIPTOR.PREVIOUS_ENUM_LIT := LAST_ENUM_LIT;
    LAST_ENUM_LIT := NEW_ENUM_LIT_DESCRIPTOR;
  end ADD_ENUM_LIT_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- ADD_FULL_ENUM_LIT_DESCRIPTOR
--
-- if this is the first full type descriptor for this enumeration literal
--             set the first pointer
-- otherwise set the "next" pointer in the previously last full enumeration
--           literal to point to this new full enumeration literal
-- set the previous pointer in this new full enumeration literal to point to
--           the old last full enumeration literal in the chain
-- and now the new full enumeration literal is the last one for this
--           enumeration literal
```

```
  procedure ADD_FULL_ENUM_LIT_DESCRIPTOR
      (NEW_FULL_ENUM_LIT_DESCRIPTOR : in out ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
       OUR_ENUM_LIT_DESCRIPTOR      : in out ACCESS_ENUM_LIT_DESCRIPTOR) is
  begin
    if OUR_ENUM_LIT_DESCRIPTOR.LAST_FULL_ENUM_LIT  = null then
      OUR_ENUM_LIT_DESCRIPTOR.FIRST_FULL_ENUM_LIT :=
                     NEW_FULL_ENUM_LIT_DESCRIPTOR;
    else
      OUR_ENUM_LIT_DESCRIPTOR.LAST_FULL_ENUM_LIT.NEXT_LIT :=
                                             NEW_FULL_ENUM_LIT_DESCRIPTOR;
    end if;
    NEW_FULL_ENUM_LIT_DESCRIPTOR.PREVIOUS_LIT :=
                             OUR_ENUM_LIT_DESCRIPTOR.LAST_FULL_ENUM_LIT;
    OUR_ENUM_LIT_DESCRIPTOR.LAST_FULL_ENUM_LIT := NEW_FULL_ENUM_LIT_DESCRIPTOR;
  end ADD_FULL_ENUM_LIT_DESCRIPTOR;

end ADD_DESCRIPTOR_ROUTINES;
```

3.11.38 package unquals.ada

```
-- unquals.ada - post process/info for unqualified names (tables & columns)

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package UNQUALIFIED_NAME is

-- Five different types of functions are generated for unqualified names:
--
-- (1) Returning TABLE_NAME for the second and subsequent table name in a list
--     of table names, and other contexts where only a single table name is
--     allowed
--
-- (2) Returning TABLE_LIST for the first table name in a list of table names
--
-- (3) Returning TABLE_NAME_WITH_COLUMN_LIST and taking a list of columns as a
--     parameter in appropriate contexts (e.g., insert column list)
--
-- (4) Returning SQL_OBJECT for column references where the result is not
--     strongly typed
--
-- (5) Returning the appropriate database type for column references where the
--     result is strongly typed

-- For uses (1) to (4), it is sufficient to maintain a list of names of
-- functions to be generated, with flags indicating whether or not each
-- particular form should be generated.  This is the purpose of the
-- UNQUALIFIED_NAME_LIST.

-- With respect to use (5), however, the same name can be used for several
-- different columns, each of which can be of a different type.  Consequently,
```

```
-- a list of return types is required for each name.  This is the purpose of
-- the RETURN_TYPE_LIST in each entry on the UNQUALIFIED_NAME_LIST.  The
-- return type is indicated by pointing to the appropriate ACCESS_FULL_NAME_-
-- DESCRIPTOR for the type.

-- See the package body for details on the data structures; visible routines
-- adjust the data structures to remember which functions must be generated.

  procedure RETURNS_TABLE_NAME ( NAME : DDL_DEFINITIONS.TYPE_NAME );

  procedure RETURNS_TABLE_LIST ( NAME : DDL_DEFINITIONS.TYPE_NAME );

  procedure RETURNS_TABLE_NAME_WITH_COLUMN_LIST
            ( NAME : DDL_DEFINITIONS.TYPE_NAME );

  procedure RETURNS_SQL_OBJECT ( NAME : DDL_DEFINITIONS.TYPE_NAME );

  procedure RETURNS_TYPED_RESULT
            ( FUNCTION_NAME : DDL_DEFINITIONS.TYPE_NAME;
              RETURN_TYPE   : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- Post processing to generate functions for the unqualified names must be
-- done in two parts:

-- (1) For each name, package name_NAME is instantiated from
--     ADA_SQL_FUNCTIONS.NAME_PACKAGE.  This is done inside a package nested
--     within the generated package, so that the instantiated packages are not
--     directly visible from the generated package.

-- (2) Each required function (name and return type) is instantiated from
--     the appropriate name_NAME package generated in (1).  The functions are
--     produced directly within the generated package for direct visibility.

-- See the package body for details on code generated; visible routines cause
-- post processing steps (1) and (2) to be performed.

  procedure POST_PROCESSING_1;

  procedure POST_PROCESSING_2;

end UNQUALIFIED_NAME;
```

3.11.39 package unqualb.ada

```
-- unqualb.ada - post process/info for unqualified names (tables & columns)

with TEXT_PRINT, DUMMY, DATABASE_TYPE;
 use TEXT_PRINT;
package body UNQUALIFIED_NAME is
```

```
type UNQUALIFIED_NAME_ENTRY_RECORD;
type RETURN_TYPE_ENTRY_RECORD;

type UNQUALIFIED_NAME_ENTRY is access UNQUALIFIED_NAME_ENTRY_RECORD;
type RETURN_TYPE_ENTRY      is access RETURN_TYPE_ENTRY_RECORD;

type RETURN_TYPE_ENTRY_RECORD is
  record
    FULL_NAME_DESCRIPTOR :
     DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
      DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
    NEXT_RETURN_TYPE :
     RETURN_TYPE_ENTRY;
  end record;

type UNQUALIFIED_NAME_ENTRY_RECORD is
  record
    NAME :
     DDL_DEFINITIONS.TYPE_NAME := DUMMY.TYPE_NAME;
    RETURNS_TABLE_NAME :
     BOOLEAN := FALSE;
    RETURNS_TABLE_LIST :
     BOOLEAN := FALSE;
    RETURNS_TABLE_NAME_WITH_COLUMN_LIST :
     BOOLEAN := FALSE;
    RETURNS_SQL_OBJECT :
     BOOLEAN := FALSE;
    RETURN_TYPE_LIST :
     RETURN_TYPE_ENTRY := new RETURN_TYPE_ENTRY_RECORD;
    NEXT_FUNCTION :
     UNQUALIFIED_NAME_ENTRY;
  end record;

UNQUALIFIED_NAME_LIST : UNQUALIFIED_NAME_ENTRY :=
 new UNQUALIFIED_NAME_ENTRY_RECORD;

function NEW_FUNCTION_NAME ( NAME : DDL_DEFINITIONS.TYPE_NAME )
 return UNQUALIFIED_NAME_ENTRY is
  CURRENT_FUNCTION :
   UNQUALIFIED_NAME_ENTRY := UNQUALIFIED_NAME_LIST;
  NEW_FUNCTION :
   UNQUALIFIED_NAME_ENTRY;
begin
  while CURRENT_FUNCTION.NEXT_FUNCTION /= null and then
   NAME.all >= CURRENT_FUNCTION.NEXT_FUNCTION.NAME.all loop
    CURRENT_FUNCTION := CURRENT_FUNCTION.NEXT_FUNCTION;
  end loop;
  if NAME.all = CURRENT_FUNCTION.NAME.all then
    return CURRENT_FUNCTION;
```

```
  else
    NEW_FUNCTION := new UNQUALIFIED_NAME_ENTRY_RECORD;
    NEW_FUNCTION.NAME := NAME;
    NEW_FUNCTION.NEXT_FUNCTION := CURRENT_FUNCTION.NEXT_FUNCTION;
    CURRENT_FUNCTION.NEXT_FUNCTION := NEW_FUNCTION;
    return NEW_FUNCTION;
  end if;
end NEW_FUNCTION_NAME;

procedure RETURNS_TABLE_NAME ( NAME : DDL_DEFINITIONS.TYPE_NAME ) is
  OUR_FUNCTION : UNQUALIFIED_NAME_ENTRY := NEW_FUNCTION_NAME ( NAME );
begin
  OUR_FUNCTION.RETURNS_TABLE_NAME := TRUE;
end RETURNS_TABLE_NAME;

procedure RETURNS_TABLE_LIST ( NAME : DDL_DEFINITIONS.TYPE_NAME ) is
  OUR_FUNCTION : UNQUALIFIED_NAME_ENTRY := NEW_FUNCTION_NAME ( NAME );
begin
  OUR_FUNCTION.RETURNS_TABLE_LIST := TRUE;
end RETURNS_TABLE_LIST;

procedure RETURNS_TABLE_NAME_WITH_COLUMN_LIST
          ( NAME : DDL_DEFINITIONS.TYPE_NAME ) is
  OUR_FUNCTION : UNQUALIFIED_NAME_ENTRY := NEW_FUNCTION_NAME ( NAME );
begin
  OUR_FUNCTION.RETURNS_TABLE_NAME_WITH_COLUMN_LIST := TRUE;
end RETURNS_TABLE_NAME_WITH_COLUMN_LIST;

procedure RETURNS_SQL_OBJECT ( NAME : DDL_DEFINITIONS.TYPE_NAME ) is
  OUR_FUNCTION : UNQUALIFIED_NAME_ENTRY := NEW_FUNCTION_NAME ( NAME );
begin
  OUR_FUNCTION.RETURNS_SQL_OBJECT := TRUE;
end RETURNS_SQL_OBJECT;

function ">=" ( LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR )
 return BOOLEAN is
begin
  if LEFT.SCHEMA_UNIT.NAME.all > RIGHT.SCHEMA_UNIT.NAME.all then
    return TRUE;
  elsif LEFT.SCHEMA_UNIT /= RIGHT.SCHEMA_UNIT then
    return FALSE;
  elsif LEFT.NAME.all >= RIGHT.NAME.all then
    return TRUE;
  else
    return FALSE;
  end if;
end ">=";

procedure RETURNS_TYPED_RESULT
```

```
          ( FUNCTION_NAME : DDL_DEFINITIONS.TYPE_NAME;
            RETURN_TYPE   : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR ) is
  CURRENT_RETURN_TYPE :
   RETURN_TYPE_ENTRY :=
    NEW_FUNCTION_NAME ( FUNCTION_NAME ) . RETURN_TYPE_LIST;
begin
  while CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE /= null and then
   RETURN_TYPE >= CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE.FULL_NAME_DESCRIPTOR
  loop
    CURRENT_RETURN_TYPE := CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE;
  end loop;
  if RETURN_TYPE /= CURRENT_RETURN_TYPE.FULL_NAME_DESCRIPTOR then
    CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE :=
     new RETURN_TYPE_ENTRY_RECORD'
     ( RETURN_TYPE, CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE );
    DATABASE_TYPE.REQUIRED_FOR ( RETURN_TYPE );
  end if;
end RETURNS_TYPED_RESULT;

procedure POST_PROCESSING_1 is
  CURRENT_FUNCTION :
   UNQUALIFIED_NAME_ENTRY := UNQUALIFIED_NAME_LIST.NEXT_FUNCTION;
begin
  if CURRENT_FUNCTION /= null then
  while CURRENT_FUNCTION /= null loop
    SET_INDENT (4);
    PRINT ( "package " );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME" );
    PRINT ( " is new");
    PRINT_LINE;
    SET_INDENT (6);
    PRINT ("ADA_SQL_FUNCTIONS.NAME_PACKAGE");
    PRINT ( "( " );
    PRINT ( """" & STRING ( CURRENT_FUNCTION.NAME.all ) & """" );
    PRINT ( " );" );
    PRINT_LINE;
    CURRENT_FUNCTION := CURRENT_FUNCTION.NEXT_FUNCTION;
  end loop;
  BLANK_LINE;
  end if;
end POST_PROCESSING_1;

procedure POST_PROCESSING_2 is
  CURRENT_FUNCTION :
   UNQUALIFIED_NAME_ENTRY := UNQUALIFIED_NAME_LIST.NEXT_FUNCTION;
  CURRENT_RETURN_TYPE :
   RETURN_TYPE_ENTRY;
begin
  if CURRENT_FUNCTION /= null then
```

```
while CURRENT_FUNCTION /= null loop
  if CURRENT_FUNCTION.RETURNS_TABLE_NAME then
    SET_INDENT (2);
    PRINT ( "function " );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) );
    PRINT ( " is new" );
    PRINT_LINE;
    SET_INDENT (4);
    PRINT ( "ADA_SQL." );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME." );
    PRINT ( "COLUMN_OR_TABLE_NAME");
    PRINT_LINE;
    SET_INDENT (6);
    PRINT ("( ADA_SQL_FUNCTIONS.TABLE_NAME );" );
    PRINT_LINE;
  end if;
  if CURRENT_FUNCTION.RETURNS_TABLE_LIST then
    SET_INDENT (2);
    PRINT ( "function " );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) );
    PRINT ( " is new" );
    PRINT_LINE;
    SET_INDENT (4);
    PRINT ( "ADA_SQL." );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME." );
    PRINT ( "COLUMN_OR_TABLE_NAME");
    PRINT_LINE;
    SET_INDENT (6);
    PRINT ("( ADA_SQL_FUNCTIONS.TABLE_LIST );" );
    PRINT_LINE;
  end if;
  if CURRENT_FUNCTION.RETURNS_TABLE_NAME_WITH_COLUMN_LIST then
    SET_INDENT (2);
    PRINT ( "function " );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) );
    PRINT ( " is new" );
    PRINT_LINE;
    SET_INDENT (4);
    PRINT ( "ADA_SQL." );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME." );
    PRINT ( "TABLE_NAME_WITH_COLUMN_LIST;" );
    PRINT_LINE;
  end if;
  if CURRENT_FUNCTION.RETURNS_SQL_OBJECT then
    SET_INDENT (2);
    PRINT ( "function " );
    PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) );
    PRINT ( " is new" );
    PRINT_LINE;
```

```
            SET_INDENT (4);
            PRINT ( "ADA_SQL." );
            PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME." );
            PRINT ( "COLUMN_OR_TABLE_NAME");
            PRINT_LINE;
            SET_INDENT (6);
            PRINT ("( ADA_SQL_FUNCTIONS.SQL_OBJECT );" );
            PRINT_LINE;
          end if;
          CURRENT_RETURN_TYPE :=
           CURRENT_FUNCTION.RETURN_TYPE_LIST.NEXT_RETURN_TYPE;
          while CURRENT_RETURN_TYPE /= null loop
            SET_INDENT (2);
            PRINT ( "function " );
            PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) );
            PRINT ( " is new");
            PRINT_LINE;
            SET_INDENT (4);
            PRINT ("ADA_SQL." );
            PRINT ( STRING ( CURRENT_FUNCTION.NAME.all ) & "_NAME." );
            PRINT ( "COLUMN_OR_TABLE_NAME");
            PRINT_LINE;
            SET_INDENT (6);
            PRINT ("( ADA_SQL." );
            PRINT
            ( STRING
              ( CURRENT_RETURN_TYPE.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT.NAME.all )
              & "_TYPE_PACKAGE." );
            PRINT
            ( STRING
              ( CURRENT_RETURN_TYPE.FULL_NAME_DESCRIPTOR.NAME.all ) & "_TYPE" );
            PRINT ( " );" );
            PRINT_LINE;
            CURRENT_RETURN_TYPE := CURRENT_RETURN_TYPE.NEXT_RETURN_TYPE;
          end loop;
          CURRENT_FUNCTION := CURRENT_FUNCTION.NEXT_FUNCTION;
        end loop;
        BLANK_LINE;
        end if;
      end POST_PROCESSING_2;

    end UNQUALIFIED_NAME;
```

3.11.40 package quals.ada

```
-- quals.ada - post process data structures for qualified column specs

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package QUALIFIED_NAME is
```

```
-- A column specification containing a qualifier that is a table name, such as
-- EMPLOYEE.JOB, returns an object of type SQL_OBJECT or of a strongly typed
-- database type (see dbtypes.ada for a discussion of strongly typed database
-- types).  This is implemented by having a function for the table name (e.g.,
-- EMPLOYEE) return an object of a record type, with components named
-- according to the columns to be selected.  Thus, EMPLOYEE.JOB selects the
-- JOB component from the result of the EMPLOYEE function, which contains the
-- required values to designate the JOB column of the EMPLOYEE table at
-- runtime.

-- We generate two versions of the function for the table name (e.g.,
-- EMPLOYEE), one returning a record with components of type SQL_OBJECT and
-- one returning a record with components of the appropriate strongly typed
-- database type for each column.  (Actually, functions are only generated as
-- required.  So either version may be generated without the other, according
-- to the column specifications found in the source program.)

-- Basically, there are three steps in defining each function for a table
-- name:
--
-- (1) Declare the record type that will be returned by the function
--
-- (2) Declare the constant object the value of which will be returned by the
--     function
--
-- (3) Instantiate CONSTANT_LITERAL to create the required function returning
--     the value (2) of type (1)

-- The record type for a table with columns named c1, c2, ..., used in
-- qualified column specifications in contexts where a untyped return data
-- structure is required, looks like:
--
--   type UNTYPED_TABLE_TYPE is
--     record
--       c1 : ADA_SQL_FUNCTIONS.SQL_OBJECT;
--       c2 : ADA_SQL_FUNCTIONS.SQL_OBJECT;
--       ...
--     end record;

-- Note that the table name does not appear in the declaration; all
-- declarations for a particular table are placed in a package specific to
-- that table, in order to avoid column name clashes.  (Items generated for
-- columns that could cause name clashes are discussed later.)

-- The similar declaration for columns c3, c4, ..., used in qualified column
-- specifications in contexts where strongly typed return data structures are
-- required, is: (b3 and b4 are the simple names of the base program types of
-- c3 and c4, respectively, and p3 and p4 are the simple names of the library
-- units in which b3 and b4 are declared.  dbtypes.ada describes how the type
```

```
-- declarations for p3_TYPE_PACKAGE.b3_TYPE and p4_TYPE_PACKAGE.b4_TYPE are
-- generated.)
--
--   type TYPED_TABLE_TYPE is
--     record
--       c3 : p3_TYPE_PACKAGE.b3_TYPE;
--       c4 : p4_TYPE_PACKAGE.b4_TYPE;
--       ...
--     end record;

-- The constant objects of these types (values to be returned by the table
-- name functions), are declared as:
--
--   UNTYPED_TABLE:
--    constant UNTYPED_TABLE_TYPE :=
--     ( c1 => c1_FUNCTION,
--       c2 => c2_FUNCTION,
--       ... );
--
--   TYPED_TABLE:
--    constant TYPED_TABLE_TYPE :=
--     ( c3 => c3_FUNCTION,
--       c4 => c4_FUNCTION,
--       ... );

-- We will come back to the declarations of the _FUNCTIONs.  If the above
-- declarations had been made for table t, which had been declared in source
-- library unit u, then the instantiations of CONSTANT_LITERAL generated to
-- actually declare the functions for the table name t used as a qualifier in
-- a column specification would be:
--
--   function t is new
--    ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
--    ( ADA_SQL.u_NAMES_PACKAGE.t_TABLE.UNTYPED_TABLE_TYPE,
--      ADA_SQL.u_NAMES_PACKAGE.t_TABLE.UNTYPED_TABLE );
--
--   function t is new
--    ADA_SQL_FUNCTIONS.CONSTANT_LITERAL
--    ( ADA_SQL.u_NAMES_PACKAGE.t_TABLE.TYPED_TABLE_TYPE,
--      ADA_SQL.u_NAMES_PACKAGE.t_TABLE.TYPED_TABLE );

-- All the qualification used to get down to the types and constants shown
-- here is indicative of the package structure used in the generated code:

-- (1) All this stuff (except for the instantiations of CONSTANT_LITERAL) is
--     placed in a package, ADA_SQL, nested within the generated package, so
--     that it will not be directly visible to the application program.  The
--     instantiations of CONSTANT_LITERAL are directly visible in the
--     generated package.
```

```
-- (2) Within the ADA_SQL package, there is one package for each DDL library
--     unit (u in the notation used in the above example) declaring tables
--     used in the source program as column specification qualifiers.  Code
--     generated for each library unit is segregated in this fashion to
--     prevent name clashes from tables with identical names being declared in
--     different DDL library units.  (This cannot occur with the current
--     implementation, which does not include authorization identifiers.)
--
-- (3) Within the package for each libary unit, there is one package for each
--     table (t in the notation of the above example) used to qualify a column
--     specification.  Code generated for each table is segregated in this
--     fashion to prevent name clashes from columns with identical names being
--     declared in different tables.

-- The _FUNCTIONs used to set values for the components of the UNTYPED_TABLE
-- and TYPED_TABLE constants for table t are created as follows:

-- (1) For each column c3, a package containing the generic function necessary
--     to produce the appropriate c3_FUNCTIONs is itself instantiated (the
--     parameter provides the string representation of the column
--     specification, which may be passed to the underlying database
--     management system at runtime):
--
--       package c3_NAME is new
--        ADA_SQL_FUNCTIONS.NAME_PACKAGE ( "t.c3" );
--
-- (2) The c3_FUNCTIONs, as required for either an SQL_OBJECT result type or a
--     strongly typed result type, or both, are then instantiated from a
--     generic function defined in c3_NAME (p3 and b3 are as used above):
--
--       function c3_FUNCTION is new
--        c3_NAME.COLUMN_OR_TABLE_NAME ( ADA_SQL_FUNCTIONS.SQL_OBJECT );
--
--       function c3_FUNCTION is new
--        c3_NAME.COLUMN_OR_TABLE_NAME
--        ( p3_TYPE_PACKAGE.b3_TYPE );

-- Considering the above, the following information is stored for each
-- different column specification used that is qualified with a table name:
--
-- (1) The identity of the package in which the table is declared
--
-- (2) The identity of the table
--
-- (3) The identity of the column
--
-- (4) The identity of the program type of the column
--
-- (5) The identity of the package in which the program type of the column is
```

```
--     declared
--
-- (6) Whether or not the column specification is to produce a result of type
--     SQL_OBJECT
--
-- (7) Whether or not the column specification is to produce a result of a
--     strongly typed database type

-- Information items (1) - (5) can all be traced from the ACCESS_FULL_NAME_-
-- DESCRIPTOR for a particular column, so that is the data structure that is
-- passed to the routines that record the appearance of each different column
-- specification.  (The calling routine verifies that the column specification
-- is valid in its context, thereby obtaining the appropriate ACCESS_FULL_-
-- NAME_DESCRIPTOR pointer.)  Which routines are called determines information
-- items (6) and (7), with the obvious meaning for QUALIFIED_NAME.RETURNS_-
-- SQL_OBJECT and QUALIFIED_NAME.RETURNS_STRONGLY_TYPED:

  procedure RETURNS_SQL_OBJECT
            ( COLUMN : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

  procedure RETURNS_STRONGLY_TYPED
            ( COLUMN : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- Post processing for column specifications qualified with a table name is
-- done in two parts:

-- (1) Produce everything except the final instantiations of CONSTANT_LITERAL:
--     Within the ADA_SQL package, generate:
--     For each relevant package declaring a table, generate a nested package:
--       For each relevant table, generate a nested package:
--          Instantiate the c_NAME packages for all relevant columns
--          Instantiate the c_FUNCTIONs for all relevant columns, with SQL_-
--           OBJECT and/or strongly typed results, as required
--          Generate the TYPED_TABLE_TYPE and TYPED_TABLE constant
--          Generate the UNTYPED_TABLE_TYPE and UNTYPED_TABLE constant
--
-- (2) Produce the instantiations of CONSTANT_LITERAL directly within the
--     generated package

-- QUALIFIED_NAME.POST_PROCESSING_1 and QUALIFIED_NAME.POST_PROCESSING_2 are
-- called to perform steps (1) and (2), respectively:

  procedure POST_PROCESSING_1;

  procedure POST_PROCESSING_2;

-- The data structure (not visible to calling routine; see package body for
-- details) used to store the required information parallels the nested
-- looping requirement of post processing step (1);
```

```
--
-- (1) A listhead points to a chain of entries, one entry for each relevant
--     package.
--
-- (2) The entry for each relevant package points to a chain of entries, one
--     entry for each relevant table.
--
-- (3) The entry for each relevant table points to a chain of entries, one
--     entry for each relevant column (the actual repository for most of the
--     information stored in the data structure).

end QUALIFIED_NAME;
```

3.11.41 package qualb.ada

```
-- qualb.ada - post process data structures for qualified column specs

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, DATABASE_TYPE;
use  TEXT_PRINT;
package body QUALIFIED_NAME is

   use DDL_DEFINITIONS;

   type QUALIFIED_NAME_ENTRY_RECORD;
   type TABLE_ENTRY_RECORD;
   type COLUMN_ENTRY_RECORD;

   type QUALIFIED_NAME_ENTRY is access QUALIFIED_NAME_ENTRY_RECORD;
   type TABLE_ENTRY is access TABLE_ENTRY_RECORD;
   type COLUMN_ENTRY is access COLUMN_ENTRY_RECORD;

   type COLUMN_ENTRY_RECORD is
      record
         COLUMN                  : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                      DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         RETURNS_SQL_OBJECT      : BOOLEAN := FALSE;
         RETURNS_STRONGLY_TYPED  : BOOLEAN := FALSE;
         NEXT_COLUMN             : COLUMN_ENTRY;
      end record;

   type TABLE_ENTRY_RECORD is
      record
         TABLE       : DDL_DEFINITIONS.RECORD_NAME := DUMMY.RECORD_NAME;
         HAS_TYPED   : BOOLEAN := FALSE;
         HAS_UNTYPED : BOOLEAN := FALSE;
         COLUMN_LIST : COLUMN_ENTRY := new COLUMN_ENTRY_RECORD;
         NEXT_TABLE  : TABLE_ENTRY;
      end record;

   type QUALIFIED_NAME_ENTRY_RECORD is
```

```
      record
         PACKAGE_NAME : DDL_DEFINITIONS.LIBRARY_UNIT_NAME :=
                           DUMMY.LIBRARY_UNIT_NAME;
         TABLE_LIST   : TABLE_ENTRY := new TABLE_ENTRY_RECORD;
         NEXT_PACKAGE : QUALIFIED_NAME_ENTRY;
      end record;

   QUALIFIED_NAME_LIST : QUALIFIED_NAME_ENTRY := new QUALIFIED_NAME_ENTRY_RECORD;

function NEW_PACKAGE
   (PACKAGE_NAME : DDL_DEFINITIONS.LIBRARY_UNIT_NAME)
   return QUALIFIED_NAME_ENTRY is
   TRACER : QUALIFIED_NAME_ENTRY := QUALIFIED_NAME_LIST;
   RESULT : QUALIFIED_NAME_ENTRY;
begin
   while TRACER.NEXT_PACKAGE /= null and then
      PACKAGE_NAME.all >= TRACER.NEXT_PACKAGE.PACKAGE_NAME.all loop
      TRACER := TRACER.NEXT_PACKAGE;
   end loop;
   if PACKAGE_NAME.all = TRACER.PACKAGE_NAME.all then
      RESULT := TRACER;
   else
      RESULT := new QUALIFIED_NAME_ENTRY_RECORD;
      RESULT.PACKAGE_NAME := PACKAGE_NAME;
      RESULT.NEXT_PACKAGE := TRACER.NEXT_PACKAGE;
      TRACER.NEXT_PACKAGE := RESULT;
   end if;
   return RESULT;
end NEW_PACKAGE;

function NEW_TABLE
   (PACKAGE_NAME        : DDL_DEFINITIONS.LIBRARY_UNIT_NAME;
    TABLE_NAME          : DDL_DEFINITIONS.RECORD_NAME;
    ADDING_TYPED_COLUMN : BOOLEAN)
   return TABLE_ENTRY is
   TRACER : TABLE_ENTRY := NEW_PACKAGE(PACKAGE_NAME).TABLE_LIST;
   RESULT : TABLE_ENTRY;
begin
   while TRACER.NEXT_TABLE /= null and then
      TABLE_NAME.all >= TRACER.NEXT_TABLE.TABLE.all loop
      TRACER := TRACER.NEXT_TABLE;
   end loop;
   if TABLE_NAME.all = TRACER.TABLE.all then
      RESULT := TRACER;
   else
      RESULT := new TABLE_ENTRY_RECORD;
      RESULT.TABLE := TABLE_NAME;
      RESULT.NEXT_TABLE := TRACER.NEXT_TABLE;
      TRACER.NEXT_TABLE := RESULT;
```

```
   end if;
   if ADDING_TYPED_COLUMN then
      RESULT.HAS_TYPED := TRUE;
   else
      RESULT.HAS_UNTYPED := TRUE;
   end if;
   return RESULT;
end NEW_TABLE;

function NEW_COLUMN
   (PACKAGE_NAME : DDL_DEFINITIONS.LIBRARY_UNIT_NAME;
    TABLE_NAME   : DDL_DEFINITIONS.RECORD_NAME;
    COLUMN       : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    IS_TYPED     : BOOLEAN)
   return COLUMN_ENTRY is
   TRACER : COLUMN_ENTRY :=
      NEW_TABLE(PACKAGE_NAME, TABLE_NAME, IS_TYPED).COLUMN_LIST;
   RESULT : COLUMN_ENTRY;
begin
   while TRACER.NEXT_COLUMN /= null and then
      COLUMN.NAME.all >= TRACER.NEXT_COLUMN.COLUMN.NAME.all loop
      TRACER := TRACER.NEXT_COLUMN;
   end loop;
   if COLUMN.all = TRACER.COLUMN.all then
      RESULT := TRACER;
   else
      RESULT := new COLUMN_ENTRY_RECORD;
      RESULT.COLUMN := COLUMN;
      RESULT.NEXT_COLUMN := TRACER.NEXT_COLUMN;
      TRACER.NEXT_COLUMN := RESULT;
   end if;
   return RESULT;
end NEW_COLUMN;

procedure RETURNS_SQL_OBJECT
   (COLUMN : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_COLUMN : COLUMN_ENTRY :=
      NEW_COLUMN (COLUMN.SCHEMA_UNIT.NAME, COLUMN.TABLE_NAME, COLUMN,
                  IS_TYPED => FALSE);
begin
   OUR_COLUMN.RETURNS_SQL_OBJECT := TRUE;
end RETURNS_SQL_OBJECT;

procedure RETURNS_STRONGLY_TYPED
   (COLUMN : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_COLUMN : COLUMN_ENTRY :=
      NEW_COLUMN (COLUMN.SCHEMA_UNIT.NAME, COLUMN.TABLE_NAME, COLUMN,
                  IS_TYPED => TRUE);
begin
```

```
      OUR_COLUMN.RETURNS_STRONGLY_TYPED := TRUE;
      DATABASE_TYPE.REQUIRED_FOR (COLUMN.TYPE_IS.BASE_TYPE.FULL_NAME);
   end RETURNS_STRONGLY_TYPED;

   procedure POST_PROCESSING_1 is
      CURRENT_PACKAGE : QUALIFIED_NAME_ENTRY := QUALIFIED_NAME_LIST.NEXT_PACKAGE;
   begin
      while CURRENT_PACKAGE /= null loop
         SET_INDENT (4);
         PRINT ("package ");
         PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) & "_NAME_PACKAGE");
         PRINT (" is");
         PRINT_LINE;
         BLANK_LINE;
         declare
            CURRENT_TABLE : TABLE_ENTRY := CURRENT_PACKAGE.TABLE_LIST.NEXT_TABLE;
         begin
            while CURRENT_TABLE /= null loop
               SET_INDENT (6);
               PRINT ("package ");
               PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE");
               PRINT (" is");
               PRINT_LINE;
               BLANK_LINE;
               declare
                  CURRENT_COLUMN : COLUMN_ENTRY :=
                                    CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
               begin
                  while CURRENT_COLUMN /= null loop
                     SET_INDENT (8);
                     PRINT ("package ");
                     PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) & "_NAME ");
                     PRINT ("is new");
                     PRINT_LINE;
                     SET_INDENT (10);
                     PRINT ("ADA_SQL_FUNCTIONS.NAME_PACKAGE");
                     PRINT_LINE;
                     SET_INDENT (12);
                     PRINT ("( ");
                     PRINT ("""" & STRING(CURRENT_TABLE.TABLE.all) & "." &
                            STRING(CURRENT_COLUMN.COLUMN.NAME.all) & """ ");
                     PRINT (");");
                     PRINT_LINE;
                     CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
                  end loop;
               end;
               BLANK_LINE;
               declare
                  CURRENT_COLUMN      : COLUMN_ENTRY :=
```

```
                                   CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
      DID_A_COLUMN : BOOLEAN;
   begin
      while CURRENT_COLUMN /= null loop
         if CURRENT_COLUMN.RETURNS_SQL_OBJECT then
            SET_INDENT (8);
            PRINT ("function ");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                   "_FUNCTION ");
            PRINT ("is new");
            PRINT_LINE;
            SET_INDENT (10);
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) & "_NAME");
            PRINT (".COLUMN_OR_TABLE_NAME");
            PRINT_LINE;
            SET_INDENT (12);
            PRINT ("( ADA_SQL_FUNCTIONS.SQL_OBJECT );");
            PRINT_LINE;
         end if;
         if CURRENT_COLUMN.RETURNS_STRONGLY_TYPED then
            SET_INDENT (8);
            PRINT ("function ");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                   "_FUNCTION ");
            PRINT ("is new");
            PRINT_LINE;
            SET_INDENT (10);
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) & "_NAME");
            PRINT (".COLUMN_OR_TABLE_NAME");
            PRINT_LINE;
            SET_INDENT (12);
            PRINT ("( ");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.
                   TYPE_IS.BASE_TYPE.FULL_NAME.SCHEMA_UNIT.NAME.all) &
                   "_TYPE_PACKAGE.");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.
                   TYPE_IS.BASE_TYPE.FULL_NAME.NAME.all) &
                   "_TYPE ");
            PRINT (");");
            PRINT_LINE;
         end if;
         CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
      end loop;
      BLANK_LINE;
      if CURRENT_TABLE.HAS_TYPED then
         SET_INDENT (8);
         PRINT ("type TYPED_TABLE_TYPE is");
         PRINT_LINE;
         SET_INDENT (10);
```

```
    PRINT ("record");
    PRINT_LINE;
    CURRENT_COLUMN := CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
    while CURRENT_COLUMN /= null loop
       if CURRENT_COLUMN.RETURNS_STRONGLY_TYPED then
          SET_INDENT (12);
          PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
          PRINT (" : ");
          PRINT (STRING(CURRENT_COLUMN.COLUMN.
                 TYPE_IS.BASE_TYPE.FULL_NAME.SCHEMA_UNIT.NAME.all)
                 "_TYPE_PACKAGE.");
          PRINT (STRING(CURRENT_COLUMN.COLUMN.
                 TYPE_IS.BASE_TYPE.FULL_NAME.NAME.all) &
                 "_TYPE;");
          PRINT_LINE;
       end if;
       CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
    end loop;
    SET_INDENT (10);
    PRINT ("end record;");
    PRINT_LINE;
    BLANK_LINE;
    SET_INDENT (8);
    PRINT ("TYPED_TABLE : constant TYPED_TABLE_TYPE :=");
    PRINT_LINE;
    SET_INDENT (10);
    PRINT ("( ");
    CURRENT_COLUMN := CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
    DID_A_COLUMN := FALSE;
    while CURRENT_COLUMN /= null loop
       if CURRENT_COLUMN.RETURNS_STRONGLY_TYPED then
          if DID_A_COLUMN then
             PRINT (",");
             PRINT_LINE;
             SET_INDENT (12);
          else
             DID_A_COLUMN := TRUE;
          end if;
          PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
          PRINT (" => ");
          PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                 "_FUNCTION");
       end if;
       CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
    end loop;
    PRINT (" );");
    PRINT_LINE;
    BLANK_LINE;
 end if;
```

```
        if CURRENT_TABLE.HAS_UNTYPED then
           SET_INDENT (8);
           PRINT ("type UNTYPED_TABLE_TYPE is");
           PRINT_LINE;
           SET_INDENT (10);
           PRINT ("record");
           PRINT_LINE;
           CURRENT_COLUMN := CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
           while CURRENT_COLUMN /= null loop
              if CURRENT_COLUMN.RETURNS_SQL_OBJECT then
                 SET_INDENT (12);
                 PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
                 PRINT (" : ADA_SQL_FUNCTIONS.SQL_OBJECT;");
                 PRINT_LINE;
              end if;
              CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
           end loop;
           SET_INDENT (10);
           PRINT ("end record;");
           PRINT_LINE;
           BLANK_LINE;
           SET_INDENT (8);
           PRINT ("UNTYPED_TABLE : constant UNTYPED_TABLE_TYPE :=");
           PRINT_LINE;
           SET_INDENT (10);
           PRINT ("( ");
           CURRENT_COLUMN := CURRENT_TABLE.COLUMN_LIST.NEXT_COLUMN;
           DID_A_COLUMN := FALSE;
           while CURRENT_COLUMN /= null loop
              if CURRENT_COLUMN.RETURNS_SQL_OBJECT then
                 if DID_A_COLUMN then
                    PRINT (",");
                    PRINT_LINE;
                    SET_INDENT (12);
                 else
                    DID_A_COLUMN := TRUE;
                 end if;
                 PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
                 PRINT (" => ");
                 PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                        "_FUNCTION");
              end if;
              CURRENT_COLUMN := CURRENT_COLUMN.NEXT_COLUMN;
           end loop;
           PRINT (" );");
           PRINT_LINE;
           BLANK_LINE;
        end if;
     end;
```

```
                  SET_INDENT (6);
                  PRINT ("end ");
                  PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE;");
                  PRINT_LINE;
                  BLANK_LINE;
                  CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
               end loop;
            end;
            SET_INDENT (4);
            PRINT ("end ");
            PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) &"_NAME_PACKAGE;");
            PRINT_LINE;
            BLANK_LINE;
            CURRENT_PACKAGE := CURRENT_PACKAGE.NEXT_PACKAGE;
         end loop;
      end POST_PROCESSING_1;

      procedure POST_PROCESSING_2 is
         CURRENT_PACKAGE : QUALIFIED_NAME_ENTRY := QUALIFIED_NAME_LIST.NEXT_PACKAGE;
      begin
         while CURRENT_PACKAGE /= null loop
            declare
               CURRENT_TABLE : TABLE_ENTRY := CURRENT_PACKAGE.TABLE_LIST.NEXT_TABLE;
            begin
               while CURRENT_TABLE /= null loop
                  if CURRENT_TABLE.HAS_TYPED then
                     SET_INDENT (2);
                     PRINT ("function ");
                     PRINT (STRING(CURRENT_TABLE.TABLE.all));
                     PRINT (" is new");
                     PRINT_LINE;
                     SET_INDENT (4);
                     PRINT ("ADA_SQL_FUNCTIONS.CONSTANT_LITERAL");
                     PRINT_LINE;
                     SET_INDENT (6);
                     PRINT ("( ADA_SQL.");
                     PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) &
                            "_NAME_PACKAGE.");
                     PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE.");
                     PRINT ("TYPED_TABLE_TYPE, ");
                     PRINT_LINE;
                     SET_INDENT (8);
                     PRINT ("ADA_SQL.");
                     PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) &
                            "_NAME_PACKAGE.");
                     PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE.");
                     PRINT ("TYPED_TABLE );");
                     PRINT_LINE;
                  end if;
```

```
            if CURRENT_TABLE.HAS_UNTYPED then
               SET_INDENT (2);
               PRINT ("function ");
               PRINT (STRING(CURRENT_TABLE.TABLE.all));
               PRINT (" is new");
               PRINT_LINE;
               SET_INDENT (4);
               PRINT ("ADA_SQL_FUNCTIONS.CONSTANT_LITERAL");
               PRINT_LINE;
               SET_INDENT (6);
               PRINT ("( ADA_SQL.");
               PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) &
                      "_NAME_PACKAGE.");
               PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE.");
               PRINT ("UNTYPED_TABLE_TYPE,");
               PRINT_LINE;
               SET_INDENT (8);
               PRINT ("ADA_SQL.");
               PRINT (STRING(CURRENT_PACKAGE.PACKAGE_NAME.all) &
                      "_NAME_PACKAGE.");
               PRINT (STRING(CURRENT_TABLE.TABLE.all) & "_TABLE.");
               PRINT ("UNTYPED_TABLE );");
               PRINT_LINE;
            end if;
            CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
         end loop;
      end;
      CURRENT_PACKAGE := CURRENT_PACKAGE.NEXT_PACKAGE;
   end loop;
end POST_PROCESSING_2;

end QUALIFIED_NAME;
```

3.11.42 package corrs.ada

```
-- corrs.ada - internal & post process data structures for correlation names

with DDL_DEFINITIONS, DUMMY;
package CORRELATION is

-- Two data structures are used to process correlation names:

-- (1) For each correlation name declaration encountered, we remember:
--
--     (a) The correlation name
--
--     (b) The identity of the table for which it is declared

-- (2) For each table referenced by one or more correlation names, we
--     remember:
```

```
--
--      (a) The identity of the table (stored as a pointer to the appropriate
--          ACCESS_TYPE_DESCRIPTOR)
--
--      (b) Whether or not a table reference in a from list requires a return
--          type of TABLE_LIST (first or only reference in from list)
--
--      (c) Whether or not a table reference in a from list requires a return
--          type of TABLE_NAME (second or subsequent reference in from list)
--
--      (d) The following information for each column of the table that appears
--          in a column specification with a correlation name qualifier:
--
--          (1) The identity of the column (stored as a pointer to the
--              appropriate ACCESS_FULL_NAME_DESCRIPTOR)
--
--          (2) Whether or not any column specification with a correlation name
--              qualifier requires a return type of SQL_OBJECT
--
--          (3) Whether or not any column specification with a correlation name
--              name qualifier requires a strongly typed database return type
--              (the appropriate type is deduced from the information in the
--              ACCESS_FULL_NAME_DESCRIPTOR)

-- Data structure (1) is used to verify that each correlation name used is
-- declared exactly once and that all uses of each correlation name (in from
-- lists) refer to the correct table.  The CORRELATION.NAME_DECLARED_LIST (see
-- package body), with entries of type CORRELATION.NAME_DECLARED_ENTRY,
-- implements data structure (1).

-- Data structure (2) is used at post process time to produce the required
-- code in the generated package.  The CORRELATION.TABLE_LIST (see package
-- body), with entries of type CORRELATION.TABLE_ENTRY, implements data
-- structure (2).

-- Data structure (2) is adjusted as a select statement is processed.  The
-- entry on the CORRELATION.TABLE_LIST that is affected for each correlation
-- name is determined by the table for which that correlation name is
-- declared.  For this reason, information item (1b) is actually stored as a
-- pointer to the appropriate entry in data structure (2).  The appropriate
-- entry for each table reference involving a correlation name is determined
-- as the from list is processed, and this information is carried in the from
-- list data structure for the duration of processing the select statement.

-- The text of a correlation name is stored as (see Group 4 operations):

  type NAME_STRING is new STRING;

  type NAME is access NAME_STRING;
```

```
-- Each entry in data structure (1) is of the following form (pointer to
-- appropriate CORRELATION.NAME_DECLARED_ENTRY is carried in the from list
-- data structures, and used by many of the routines below):

  type NAME_DECLARED_ENTRY_RECORD is private;

  type NAME_DECLARED_ENTRY is access NAME_DECLARED_ENTRY_RECORD;

-- Group 1 operations: Called when a correlation name declaration is
-- encountered

  -- CORRELATION.NAME_DECLARATION_IS_VALID
  -- Add a correlation name declaration to data structure (1), creating data
  --  structure (2) entry for table if not already existing
  -- Called with:
  --   String representation of correlation name declared
  --   ACCESS_TYPE_DESCRIPTOR for the referenced table (validated by calling
  --    routine)
  -- Returns:
  --   TRUE if correlation name declaration is valid
  --   FALSE on error (correlation name already defined)

  function NAME_DECLARATION_IS_VALID
          ( CORRELATION_NAME : STRING;
            TABLE            : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR )
   return BOOLEAN;

-- Group 2 operations: Called when a correlation name is encountered in a
-- from list

  -- If the table reference being processed is the first in a from list, then
  -- CORRELATION.NAME_RETURNS_TABLE_LIST is called.  If the table reference
  -- being processed is the second or subsequent one in a from list, then
  -- CORRELATION.NAME_RETURNS_TABLE_NAME is called.  These set the appropriate
  -- flag (b or c) in data structure (2).  They also return the pointer to the
  -- entry for the correlation name in data structure (1), for use during
  -- following processing.  Specific parameters are:
  --   (in)  String representation of correlation name used
  --   (in)  ACCESS_TYPE_DESCRIPTOR for the referenced table (validated by
  --         calling routine)
  --   (out) Status code, as indicated below (calling routine responsible for
  --         reporting error and skipping rest of statement, and also for
  --         verifying that correlation name is not exposed elsewhere in from
  --         list)
  --   (out) If status is CORRELATION.NAME_VALID, pointer to appropriate data
  --         structure (1) entry

  type NAME_REFERENCE_STATUS is
   ( NAME_VALID,
```

```
      NAME_NOT_DECLARED,
      NAME_DECLARED_FOR_DIFFERENT_TABLE );

   procedure NAME_RETURNS_TABLE_LIST
             ( CORRELATION_NAME : in  STRING;
               TABLE            : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
               STATUS           : out NAME_REFERENCE_STATUS;
               NAME_DECLARED    : out NAME_DECLARED_ENTRY );

   procedure NAME_RETURNS_TABLE_NAME
             ( CORRELATION_NAME : in  STRING;
               TABLE            : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
               STATUS           : out NAME_REFERENCE_STATUS;
               NAME_DECLARED    : out NAME_DECLARED_ENTRY );

-- Group 3 operations: Called when the return type required for a column
-- specification including a correlation name qualifier has been determined

   -- If the column specification is to return an untyped result, then
   -- CORRELATION.COLUMN_RETURNS_SQL_OBJECT is called.  If the column
   -- specification is to return a strongly typed result, then CORRELATION.-
   -- COLUMN_RETURNS_STRONGLY_TYPED is called.  These set the appropriate flags
   -- (2 or 3) in data structure (2d), making an entry for the column if one
   -- does not already exist.  Specific parameters:
   --   Data structure (1) pointer for the correlation name used as the
   --    qualifier in the column specification
   --   ACCESS_FULL_NAME_DESCRIPTOR for the column specified (calling routine
   --    validates that column is indeed in the table designated by the
   --    correlation name)

   procedure COLUMN_RETURNS_SQL_OBJECT
             ( CORRELATION_NAME :
                NAME_DECLARED_ENTRY;
               COLUMN :
                DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

   procedure COLUMN_RETURNS_STRONGLY_TYPED
             ( CORRELATION_NAME :
                NAME_DECLARED_ENTRY;
               COLUMN :
                DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- Group 4 operations: Called to get text of correlation name when looking
-- through the from list to (1) verify that an exposed name is not
-- duplicated, or (2) verify that a qualifier used in a column specification
-- is exposed in the from list.

   -- Called with: Pointer to data structure (1) entry (stored in the from
   --  list data structure)
```

```
  -- Returns: Text of associated correlation name

  function NAME_DECLARED_FOR ( CORRELATION : NAME_DECLARED_ENTRY )
   return NAME;

-- Group 5 operations: Called to get data structure (ACCESS_TYPE_DESCRIPTOR)
-- for the table designated by a given correlation name.  This is done when
-- processing a column specification containing a correlation name, to verify
-- that the named column appears in the designated table.

  -- Called with: Pointer to data structure (1) entry for the given
  --  correlation name (taken from the from list data structure)
  -- Returns: Pointer to ACCESS_TYPE_DESCRIPTOR for the appropriate table

  function TABLE_DECLARED_FOR ( CORRELATION : NAME_DECLARED_ENTRY )
   return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

-- Group 6 operations: Called when validating names, to verify that a given
-- identifier is not hidden by a correlation name declaration with the same
-- identifier.

--   Called with: String representation of the identifier in question
--   Returns: TRUE if a correlation name with that identifier has been
--    declared, FALSE otherwise

  function NAME_IS_DECLARED ( IDENTIFIER : STRING ) return BOOLEAN;

-- Group 7 operations: Called at post process time to produce the required
-- functions in the generated package.

  procedure NAME_POST_PROCESS;

  procedure NAME_POST_PROCESS_KLUDGE; -- for VAX Ada bug
private

  type COLUMN_REFERENCE_ENTRY_RECORD;

  type COLUMN_REFERENCE_ENTRY is access COLUMN_REFERENCE_ENTRY_RECORD;

  type COLUMN_REFERENCE_ENTRY_RECORD is
    record
      COLUMN :
       DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
        DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
      RETURNS_SQL_OBJECT :
       BOOLEAN := FALSE;
      RETURNS_STRONGLY_TYPED :
       BOOLEAN := FALSE;
      NEXT_REFERENCE :
```

```
        COLUMN_REFERENCE_ENTRY;
    end record;

  type TABLE_ENTRY_RECORD;

  type TABLE_ENTRY is access TABLE_ENTRY_RECORD;

  type TABLE_ENTRY_RECORD is
    record
      DESCRIPTOR :
        DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := DUMMY.ACCESS_TYPE_DESCRIPTOR;
      RETURNS_TABLE_LIST :
        BOOLEAN := FALSE;
      RETURNS_TABLE_NAME :
        BOOLEAN := FALSE;
      COLUMN_REFERENCE_LIST :
        COLUMN_REFERENCE_ENTRY := new COLUMN_REFERENCE_ENTRY_RECORD;
      NEXT_TABLE :
        TABLE_ENTRY;
    end record;

  type NAME_DECLARED_ENTRY_RECORD is
    record
      CORRELATION_NAME    : NAME := new NAME_STRING'("");
      TABLE               : TABLE_ENTRY := new TABLE_ENTRY_RECORD;
      NEXT_NAME_DECLARED : NAME_DECLARED_ENTRY;
    end record;

end CORRELATION;
```

3.11.43 package corrb.ada

```
-- corrb.ada - post process/info for correlation names

with TEXT_PRINT, DUMMY, DDL_DEFINITIONS, DATABASE_TYPE, EXTRA_DEFINITIONS;
 use TEXT_PRINT;
package body CORRELATION is

   use DDL_DEFINITIONS;

   COMPILATION_UNIT_BEING_SCANNED
      : DDL_DEFINITIONS.ACCESS_SCHEMA_UNIT_DESCRIPTOR renames
        EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT;

   NAME_DECLARED_LIST : NAME_DECLARED_ENTRY := new NAME_DECLARED_ENTRY_RECORD;
   TABLE_LIST         : TABLE_ENTRY         := NAME_DECLARED_LIST.TABLE;

function NEW_TABLE
   ( TABLE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR )
   return TABLE_ENTRY is
```

```
   CURRENT_TABLE    : TABLE_ENTRY := CORRELATION.TABLE_LIST;
   NEW_TABLE_ENTRY : TABLE_ENTRY;
begin
   while CURRENT_TABLE.NEXT_TABLE /= null and then
      TABLE.FULL_NAME.NAME.all >=
      CURRENT_TABLE.NEXT_TABLE.DESCRIPTOR.FULL_NAME.NAME.all loop
      CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
   end loop;
   if TABLE = CURRENT_TABLE.DESCRIPTOR then
      return CURRENT_TABLE;
   else
      NEW_TABLE_ENTRY := new TABLE_ENTRY_RECORD;
      NEW_TABLE_ENTRY.DESCRIPTOR := TABLE;
      NEW_TABLE_ENTRY.NEXT_TABLE := CURRENT_TABLE.NEXT_TABLE;
      CURRENT_TABLE.NEXT_TABLE := NEW_TABLE_ENTRY;
      return NEW_TABLE_ENTRY;
   end if;
end NEW_TABLE;

function NAME_DECLARATION_IS_VALID
   ( CORRELATION_NAME : STRING;
     TABLE             : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR )
   return BOOLEAN is
   CURRENT_NAME : NAME_DECLARED_ENTRY := CORRELATION.NAME_DECLARED_LIST;
   NEW_NAME     : NAME_DECLARED_ENTRY;
begin
   while CURRENT_NAME.NEXT_NAME_DECLARED /= null and then
      NAME_STRING(CORRELATION_NAME) >=
      CURRENT_NAME.NEXT_NAME_DECLARED.CORRELATION_NAME.all loop
      CURRENT_NAME := CURRENT_NAME.NEXT_NAME_DECLARED;
   end loop;
   if NAME_STRING(CORRELATION_NAME) = CURRENT_NAME.CORRELATION_NAME.all then
      return FALSE;
   else
      NEW_NAME := new NAME_DECLARED_ENTRY_RECORD;
      NEW_NAME.CORRELATION_NAME := new NAME_STRING(CORRELATION_NAME'RANGE);
      NEW_NAME.CORRELATION_NAME.all := NAME_STRING(CORRELATION_NAME);
      NEW_NAME.TABLE := NEW_TABLE (TABLE);
      NEW_NAME.NEXT_NAME_DECLARED := CURRENT_NAME.NEXT_NAME_DECLARED;
      CURRENT_NAME.NEXT_NAME_DECLARED := NEW_NAME;
      return TRUE;
   end if;
end NAME_DECLARATION_IS_VALID;

function FIND_CORRELATION_NAME
   ( CORRELATION_NAME : STRING)
   return NAME_DECLARED_ENTRY is
   CURRENT_NAME : NAME_DECLARED_ENTRY := CORRELATION.NAME_DECLARED_LIST;
begin
```

```
   while CURRENT_NAME.NEXT_NAME_DECLARED /= null and then
      NAME_STRING(CORRELATION_NAME) >=
      CURRENT_NAME.NEXT_NAME_DECLARED.CORRELATION_NAME.all loop
      CURRENT_NAME := CURRENT_NAME.NEXT_NAME_DECLARED;
   end loop;
   if NAME_STRING(CORRELATION_NAME) = CURRENT_NAME.CORRELATION_NAME.all then
      return CURRENT_NAME;
   else
      return null;
   end if;
end FIND_CORRELATION_NAME;

procedure NAME_RETURNS_TABLE_LIST
   ( CORRELATION_NAME : in  STRING;
     TABLE            : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
     STATUS           : out NAME_REFERENCE_STATUS;
     NAME_DECLARED    : out NAME_DECLARED_ENTRY ) is
   CORRELATION_ENTRY : NAME_DECLARED_ENTRY :=
                         FIND_CORRELATION_NAME (CORRELATION_NAME);
begin
   if CORRELATION_ENTRY = null then
      STATUS := CORRELATION.NAME_NOT_DECLARED;
      NAME_DECLARED := null;
   elsif CORRELATION_ENTRY.TABLE.DESCRIPTOR /= TABLE then
      STATUS := CORRELATION.NAME_DECLARED_FOR_DIFFERENT_TABLE;
      NAME_DECLARED := null;
   else
      STATUS := CORRELATION.NAME_VALID;
      NAME_DECLARED := CORRELATION_ENTRY;
      CORRELATION_ENTRY.TABLE.RETURNS_TABLE_LIST := TRUE;
   end if;
end NAME_RETURNS_TABLE_LIST;

procedure NAME_RETURNS_TABLE_NAME
   ( CORRELATION_NAME : in  STRING;
     TABLE            : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
     STATUS           : out NAME_REFERENCE_STATUS;
     NAME_DECLARED    : out NAME_DECLARED_ENTRY ) is
   CORRELATION_ENTRY : NAME_DECLARED_ENTRY :=
                         FIND_CORRELATION_NAME (CORRELATION_NAME);
begin
   if CORRELATION_ENTRY = null then
      STATUS := CORRELATION.NAME_NOT_DECLARED;
      NAME_DECLARED := null;
   elsif CORRELATION_ENTRY.TABLE.DESCRIPTOR /= TABLE then
      STATUS := CORRELATION.NAME_DECLARED_FOR_DIFFERENT_TABLE;
      NAME_DECLARED := null;
   else
      STATUS := CORRELATION.NAME_VALID;
```

```
      NAME_DECLARED := CORRELATION_ENTRY;
      CORRELATION_ENTRY.TABLE.RETURNS_TABLE_NAME := TRUE;
   end if;
end NAME_RETURNS_TABLE_NAME;

function NEW_COLUMN
   ( CORRELATION_NAME : NAME_DECLARED_ENTRY;
     COLUMN           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR )
   return COLUMN_REFERENCE_ENTRY is
   CURRENT_COLUMN : COLUMN_REFERENCE_ENTRY :=
                       CORRELATION_NAME.TABLE.COLUMN_REFERENCE_LIST;
   NEW_COL        : COLUMN_REFERENCE_ENTRY;
begin
   while CURRENT_COLUMN.NEXT_REFERENCE /= null and then
      COLUMN.NAME.all >= CURRENT_COLUMN.NEXT_REFERENCE.COLUMN.NAME.all loop
      CURRENT_COLUMN := CURRENT_COLUMN.NEXT_REFERENCE;
   end loop;
   if COLUMN = CURRENT_COLUMN.COLUMN then
      return CURRENT_COLUMN;
   else
      NEW_COL := new COLUMN_REFERENCE_ENTRY_RECORD;
      NEW_COL.COLUMN := COLUMN;
      NEW_COL.NEXT_REFERENCE := CURRENT_COLUMN.NEXT_REFERENCE;
      CURRENT_COLUMN.NEXT_REFERENCE := NEW_COL;
      return NEW_COL;
   end if;
end NEW_COLUMN;

procedure COLUMN_RETURNS_SQL_OBJECT
   ( CORRELATION_NAME : NAME_DECLARED_ENTRY;
     COLUMN           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR ) is
   OUR_COLUMN : COLUMN_REFERENCE_ENTRY := NEW_COLUMN (CORRELATION_NAME, COLUMN);
begin
   OUR_COLUMN.RETURNS_SQL_OBJECT := TRUE;
end COLUMN_RETURNS_SQL_OBJECT;

procedure COLUMN_RETURNS_STRONGLY_TYPED
   ( CORRELATION_NAME : NAME_DECLARED_ENTRY;
     COLUMN           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR ) is
   OUR_COLUMN : COLUMN_REFERENCE_ENTRY := NEW_COLUMN (CORRELATION_NAME, COLUMN);
begin
   OUR_COLUMN.RETURNS_STRONGLY_TYPED := TRUE;
   DATABASE_TYPE.REQUIRED_FOR (COLUMN.TYPE_IS.BASE_TYPE.FULL_NAME);
end COLUMN_RETURNS_STRONGLY_TYPED;

function NAME_DECLARED_FOR
   ( CORRELATION : NAME_DECLARED_ENTRY )
   return NAME is
begin
```

```
      return CORRELATION.CORRELATION_NAME;
   end NAME_DECLARED_FOR;

   function TABLE_DECLARED_FOR
      ( CORRELATION : NAME_DECLARED_ENTRY )
      return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
   begin
      return CORRELATION.TABLE.DESCRIPTOR;
   end TABLE_DECLARED_FOR;

   function NAME_IS_DECLARED
      ( IDENTIFIER : STRING )
      return BOOLEAN is
      CURRENT_NAME : NAME_DECLARED_ENTRY := CORRELATION.NAME_DECLARED_LIST;
   begin
      while CURRENT_NAME /= null and then
         NAME_STRING(IDENTIFIER) /= CURRENT_NAME.CORRELATION_NAME.all loop
         CURRENT_NAME := CURRENT_NAME.NEXT_NAME_DECLARED;
      end loop;
      return (CURRENT_NAME /= null);
   end NAME_IS_DECLARED;

   procedure NAME_POST_PROCESS is
      CURRENT_TABLE : TABLE_ENTRY := CORRELATION.TABLE_LIST.NEXT_TABLE;
   begin
      while CURRENT_TABLE /= null loop
         declare
            CURRENT_TABLE_NAME : DDL_DEFINITIONS.TYPE_NAME :=
                                      CURRENT_TABLE.DESCRIPTOR.FULL_NAME.NAME;
         begin
            SET_INDENT (2);
            PRINT ("package ");
            PRINT (STRING(CURRENT_TABLE_NAME.all) & "_CORRELATION ");
            PRINT ("is");
            PRINT_LINE;
            BLANK_LINE;
            SET_INDENT (4);
            PRINT ("generic");
            PRINT_LINE;
            PRINT ("  CORRELATION_NAME : in STANDARD.STRING;");
            PRINT_LINE;
            PRINT ("package NAME is");
            PRINT_LINE;
            BLANK_LINE;
            SET_INDENT (6);
            PRINT ("package ADA_SQL is");
            PRINT_LINE;
            BLANK_LINE;
            SET_INDENT (8);
```

```
PRINT ("package ");
PRINT (STRING(CURRENT_TABLE_NAME.all) & "_TABLE_NAME ");
PRINT ("is new");
PRINT_LINE;
SET_INDENT (10);
PRINT ("ADA_SQL_FUNCTIONS.NAME_PACKAGE ( ");
PRINT ("""" & STRING(CURRENT_TABLE_NAME.all) & " """);
PRINT (" & CORRELATION_NAME );");
PRINT_LINE;
declare
   CURRENT_COLUMN : COLUMN_REFERENCE_ENTRY :=
      CURRENT_TABLE.COLUMN_REFERENCE_LIST.NEXT_REFERENCE;
begin
   while CURRENT_COLUMN /= null loop
      SET_INDENT (8);
      PRINT ("package ");
      PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) & "_COLUMN_NAME ");
      PRINT ("is new");
      PRINT_LINE;
      SET_INDENT (10);
      PRINT ("ADA_SQL_FUNCTIONS.NAME_PACKAGE ( " &
             "CORRELATION_NAME & ");
      PRINT ("""." & STRING(CURRENT_COLUMN.COLUMN.NAME.all) & """");
      PRINT (" );");
      PRINT_LINE;
      CURRENT_COLUMN := CURRENT_COLUMN.NEXT_REFERENCE;
   end loop;
end;
BLANK_LINE;
SET_INDENT (6);
PRINT ("end ADA_SQL;");
PRINT_LINE;
BLANK_LINE;
if CURRENT_TABLE.RETURNS_TABLE_LIST then
   SET_INDENT (6);
   PRINT ("function ");
   PRINT (STRING(CURRENT_TABLE_NAME.all));
   PRINT (" is new");
   PRINT_LINE;
   SET_INDENT (8);
   PRINT ("ADA_SQL.");
   PRINT (STRING(CURRENT_TABLE_NAME.all) &
          "_TABLE_NAME.COLUMN_OR_TABLE_NAME ");
   PRINT_LINE;
   SET_INDENT (10);
   PRINT ("( ADA_SQL_FUNCTIONS.TABLE_LIST );");
   PRINT_LINE;
end if;
if CURRENT_TABLE.RETURNS_TABLE_NAME then
```

```
      SET_INDENT (6);
      PRINT ("function ");
      PRINT (STRING(CURRENT_TABLE_NAME.all));
      PRINT (" is new");
      PRINT_LINE;
      SET_INDENT (8);
      PRINT ("ADA_SQL.");
      PRINT (STRING(CURRENT_TABLE_NAME.all) &
             "_TABLE_NAME.COLUMN_OR_TABLE_NAME ");
      SET_INDENT (10);
      PRINT ("( ADA_SQL_FUNCTIONS.TABLE_NAME );");
      PRINT_LINE;
   end if;
   declare
      CURRENT_COLUMN : COLUMN_REFERENCE_ENTRY :=
         CURRENT_TABLE.COLUMN_REFERENCE_LIST.NEXT_REFERENCE;
   begin
      while CURRENT_COLUMN /= null loop
         if CURRENT_COLUMN.RETURNS_SQL_OBJECT then
            SET_INDENT (6);
            PRINT ("function ");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
            PRINT (" is new");
            PRINT_LINE;
            SET_INDENT (8);
            PRINT ("ADA_SQL.");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                   "_COLUMN_NAME.COLUMN_OR_TABLE_NAME ");
            PRINT_LINE;
            SET_INDENT (10);
            PRINT ("( ADA_SQL_FUNCTIONS.SQL_OBJECT );");
            PRINT_LINE;
         end if;
         if CURRENT_COLUMN.RETURNS_STRONGLY_TYPED then
            SET_INDENT (6);
            PRINT ("function ");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all));
            PRINT (" is new");
            PRINT_LINE;
            SET_INDENT (8);
            PRINT ("ADA_SQL.");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.NAME.all) &
                  "_COLUMN_NAME.COLUMN_OR_TABLE_NAME ");
            SET_INDENT (10);
            PRINT ("( ");
            PRINT (STRING(COMPILATION_UNIT_BEING_SCANNED.NAME.all) &
                   "_ADA_SQL.");
            PRINT ("ADA_SQL.");
            PRINT (STRING(CURRENT_COLUMN.COLUMN.TYPE_IS.BASE_TYPE.
```

```
                              FULL_NAME.SCHEMA_UNIT.NAME.all) &
                              "_TYPE_PACKAGE.");
                       PRINT (STRING(CURRENT_COLUMN.COLUMN.TYPE_IS.BASE_TYPE.
                              FULL_NAME.NAME.all) &
                              "_TYPE ");
                       PRINT (");");
                       PRINT_LINE;
                    end if;
                    CURRENT_COLUMN := CURRENT_COLUMN.NEXT_REFERENCE;
                 end loop;
              end;
              BLANK_LINE;
              SET_INDENT (4);
              PRINT ("end NAME;");
              PRINT_LINE;
              BLANK_LINE;
              SET_INDENT (2);
              PRINT ("end ");
              PRINT (STRING(CURRENT_TABLE_NAME.all) & "_CORRELATION;");
              PRINT_LINE;
              BLANK_LINE;
           end;
           CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
        end loop;
     end NAME_POST_PROCESS;

     procedure NAME_POST_PROCESS_KLUDGE is
        CURRENT_TABLE : TABLE_ENTRY := CORRELATION.TABLE_LIST.NEXT_TABLE;
     begin
        while CURRENT_TABLE /= null loop
           declare
              CURRENT_TABLE_NAME : DDL_DEFINITIONS.TYPE_NAME :=
                                        CURRENT_TABLE.DESCRIPTOR.FULL_NAME.NAME;
           begin
              SET_INDENT (2);
              PRINT ("package body ");
              PRINT (STRING(CURRENT_TABLE_NAME.all) & "_CORRELATION ");
              PRINT ("is");
              PRINT_LINE;
              PRINT ("  package body NAME is");
              PRINT_LINE;
              PRINT ("    VAX_ADA_BUG : ADA_SQL_FUNCTIONS.SQL_OBJECT;");
              PRINT_LINE;
              PRINT ("  end NAME;");
              PRINT_LINE;
              PRINT ("end ");
              PRINT (STRING(CURRENT_TABLE_NAME.all) & "_CORRELATION;");
              PRINT_LINE;
              BLANK_LINE;
```

```
      end;
      CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
   end loop;
end NAME_POST_PROCESS_KLUDGE;

end CORRELATION;
```

3.11.44 package convs.ada

```
-- convs.ada - post process data structure for CONVERT_TO functions

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package CONVERT_TO is

-- Ada/SQL allows explicit type conversions on database values, just as Ada
-- allows explicit type conversions on program values.  The Ada/SQL syntax for
-- this is:
--
--   CONVERT_TO.library_unit.program_type ( database_value )
--
-- where
--
--   library_unit = the library unit in which program_type is declared
--   program_type = the type to which the database value should be converted
--    (because of the requirement for a nested ADA_SQL package within each DDL
--    library unit, the program type is actually visible as library_unit.ADA_-
--    SQL.program_type, but we use a shortcut and omit the intervening ADA_SQL
--    for the CONVERT_TO syntax)
--   database_value = the database value to be converted

-- Example: Our database contains a table with one row for each division in
-- our company.  Important columns are: BEER_ON_HAND, of type BEER_CANS, tells
-- how many cans of beer each division has in their storage locker, and
-- NUMBER_OF_EMPLOYEES, of type EMPLOYEE_COUNT, tells how many employees are
-- in the division.  When we plan a picnic, we want to know which divisions
-- have to order more beer.  We know that, on the average, each employee (and
-- the guests he brings) consumes twelve cans of beer at a picnic.  The
-- following is the search condition for determining which divisions must
-- order more beer for the picnic (assuming that all types are declared in
-- DDL library unit COMPANY_TYPES):
--
--   WHERE => BEER_ON_HAND <
--              12 * CONVERT_TO.COMPANY_TYPES.BEER_CANS ( NUMBER_OF_EMPLOYEES )

-- The parameter to CONVERT_TO may be an arbitrary expression (providing the
-- types are right, of course), so the following equivalent search condition
-- is also possible:
--
--   WHERE => BEER_ON_HAND <
```

```
--              CONVERT_TO.COMPANY_TYPES.BEER_CANS ( 12 * NUMBER_OF_EMPLOYEES )

-- The parameter to CONVERT_TO must be a database value, however.  Ordinary
-- Ada type conversion is used with program values.  Example: We are running
-- an internal audit to determine which divisions are stocking up too much
-- beer.  Program variable BEER_LIMIT, of type SIX_PACKS, contains the maximum
-- number of six packs of beer that a division may retain.  The search
-- condition to determine which divisions are stocking up too much beer is:
--
--   WHERE => BEER_ON_HAND >
--              6 * COMPANY_TYPES.ADA_SQL.BEER_CANS ( BEER_LIMIT )
--
-- Note that the correctly qualified BEER_CANS type name must be used here,
-- since it is Ada, not the application scanner, that is generating the
-- necessary code (actually none for a typical Ada compiler) for the type
-- conversion.  The application scanner generates nothing for Ada type
-- conversions, but does keep track of types to verify that subsequent uses in
-- expressions are valid.

-- Since the application scanner does verify that operations are performed
-- only on comparable types, type conversions (Ada/SQL for database values,
-- Ada for program values) are required for many operations on objects of
-- different types.  The application scanner would reject the "<" and ">"
-- operators in the following search conditions:
--
--   WHERE => BEER_ON_HAND < 12 * NUMBER_OF_EMPLOYEES
--
--   WHERE => BEER_ON_HAND > 6 * BEER_LIMIT

-- The CONVERT_TO functions are generated in a package CONVERT_TO, nested
-- within the generated package.  Within the CONVERT_TO package is one package
-- (e.g., COMPANY_TYPES) for each library unit declaring target types for the
-- conversions.  The actual conversion functions for each type are generated
-- within the package produced for the library unit in which the type is
-- declared.

-- A CONVERT_TO function will return an object of type SQL_OBJECT if it is
-- used in a context where an untyped return value is required, and will
-- return an object of a strongly typed database type (see dbtypes.ada) in
-- contexts where a strongly typed return value is required.  The code
-- generated for converting to type b, declared in library unit p, for each
-- of these cases is:
--
--   function b ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
--    return ADA_SQL_FUNCTIONS.SQL_OBJECT renames CONVERT_R;
--
--   function b ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )
--    return ADA_SQL.p_TYPE_PACKAGE.b_TYPE renames CONVERT_R;
```

```
-- ADA_SQL.p_TYPE_PACKAGE.b_TYPE is declared by code generated from dbtypes.-
-- ada.  CONVERT_R is predefined returning SQL_OBJECT, and is derived
-- returning ADA_SQL.p_TYPE_PACKAGE.b_TYPE when that type is declared.

-- The information that must be known for each type used as the target of a
-- CONVERT_TO operation is:
--
-- (1) The identity of the type
--
-- (2) The identity of the library unit in which the target type is declared
--
-- (3) Whether or not a conversion function returning SQL_OBJECT should be
--     generated for the type
--
-- (4) Whether or not a conversion function returning a strongly typed result
--     should be generated for the type

-- Information items (1) and (2) can be deduced from the ACCESS_FULL_NAME_-
-- DESCRIPTOR for the type.  CONVERT_TO.RETURNS_SQL_OBJECT and CONVERT_TO.-
-- RETURNS_STRONGLY_TYPED record the fact that a CONVERT_TO function will be
-- required for the type specified by its ACCESS_FULL_NAME_DESCRIPTOR.  Which
-- routine is called determines information items (3) and (4).  (A particular
-- type may require only one CONVERT_TO function, or may require both,
-- depending on the conversions used within the source program.)  The routines
-- are called each time a CONVERT_TO function is found to be required; they
-- automatically ignore duplicate requests.

  procedure RETURNS_SQL_OBJECT
            ( TARGET_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

  procedure RETURNS_STRONGLY_TYPED
            ( TARGET_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- Post processing of the CONVERT_TO functions is:
-- Within the CONVERT_TO package, generate:
-- A package for each library unit declaring target types for conversions,
-- containing:
--   Functions, returning SQL_OBJECT, strongly typed, or both, for each target
--   type declared in that library unit

-- This processing is performed by a call to CONVERT_TO.POST_PROCESSING:

  procedure POST_PROCESSING;

-- The form of the data structure used to retain the CONVERT_TO information
-- (see package body; data structure not visible to calling routines)
-- parallels the nesting required by the post processing:
--
-- A listhead points to a chain of entries, one entry for each relevant
```

```
-- library unit, pointing to:
--   A chain of entries, one entry for each relevant type declared within that
--   library unit, and containing information items (1), (3) and (4)

end CONVERT_TO;
```

3.11.45 package convb.ada

```
-- convb.ada - post process data structure for CONVERT_TO functions

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, DATABASE_TYPE;
use  TEXT_PRINT;
package body CONVERT_TO is

   use DDL_DEFINITIONS;

   type CONVERT_TO_ENTRY_RECORD;
   type CONVERT_TO_ENTRY is access CONVERT_TO_ENTRY_RECORD;

   type CONVERT_TO_ENTRY_RECORD is
      record
         TARGET_TYPE              : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                       DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         RETURNS_SQL_OBJECT       : BOOLEAN := FALSE;
         RETURNS_STRONGLY_TYPED : BOOLEAN := FALSE;
         NEXT_TYPE                : CONVERT_TO_ENTRY;
      end record;

   CONVERT_TO_LIST : CONVERT_TO_ENTRY := new CONVERT_TO_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.SCHEMA_UNIT.NAME.all > RIGHT.SCHEMA_UNIT.NAME.all then
      return TRUE;
   elsif LEFT.SCHEMA_UNIT /= RIGHT.SCHEMA_UNIT then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";

function NEW_CONVERT_TO_TYPE
   (TARGET_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return CONVERT_TO_ENTRY is
   TRACER : CONVERT_TO_ENTRY := CONVERT_TO_LIST;
   -- Order list by fully-qualified target type name.
```

```
begin
   while TRACER.NEXT_TYPE /= null and then
         TARGET_TYPE >= TRACER.NEXT_TYPE.TARGET_TYPE loop
      TRACER := TRACER.NEXT_TYPE;
   end loop;
   if TARGET_TYPE = TRACER.TARGET_TYPE then
      return TRACER;
   else
      TRACER.NEXT_TYPE := new CONVERT_TO_ENTRY_RECORD'
                         (TARGET_TYPE             => TARGET_TYPE,
                          RETURNS_SQL_OBJECT      => FALSE,
                          RETURNS_STRONGLY_TYPED => FALSE,
                          NEXT_TYPE               => TRACER.NEXT_TYPE);
      return TRACER.NEXT_TYPE;
   end if;
end NEW_CONVERT_TO_TYPE;

procedure RETURNS_SQL_OBJECT
   (TARGET_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_ENTRY : CONVERT_TO_ENTRY := NEW_CONVERT_TO_TYPE (TARGET_TYPE);
begin
   OUR_ENTRY.RETURNS_SQL_OBJECT := TRUE;
end RETURNS_SQL_OBJECT;

procedure RETURNS_STRONGLY_TYPED
   (TARGET_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_ENTRY : CONVERT_TO_ENTRY := NEW_CONVERT_TO_TYPE (TARGET_TYPE);
begin
   OUR_ENTRY.RETURNS_STRONGLY_TYPED := TRUE;
   DATABASE_TYPE.REQUIRED_FOR (TARGET_TYPE);
end RETURNS_STRONGLY_TYPED;

procedure POST_PROCESSING is
   TRACER         : CONVERT_TO_ENTRY := CONVERT_TO_LIST.NEXT_TYPE;
   CURRENT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR;
begin
   if TRACER /= null then
      SET_INDENT (2);
      PRINT ("package CONVERT_TO is");
      PRINT_LINE;
      BLANK_LINE;
      while TRACER /= null loop
         CURRENT_SCHEMA := TRACER.TARGET_TYPE.SCHEMA_UNIT;
         SET_INDENT (4);
         PRINT ("package ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all));
         PRINT (" is");
         PRINT_LINE;
         BLANK_LINE;
```

```
         while TRACER /= null and then
            TRACER.TARGET_TYPE.SCHEMA_UNIT = CURRENT_SCHEMA loop
            if TRACER.RETURNS_SQL_OBJECT then
               SET_INDENT (6);
               PRINT ("function ");
               PRINT (STRING(TRACER.TARGET_TYPE.NAME.all));
               PRINT_LINE;
               SET_INDENT (8);
               PRINT ("( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )");
               PRINT_LINE;
               PRINT ("return  ADA_SQL_FUNCTIONS.SQL_OBJECT");
               PRINT_LINE;
               PRINT ("renames CONVERT_R;");
               PRINT_LINE;
               BLANK_LINE;
            end if;
            if TRACER.RETURNS_STRONGLY_TYPED then
               SET_INDENT (6);
               PRINT ("function ");
               PRINT (STRING(TRACER.TARGET_TYPE.NAME.all));
               PRINT_LINE;
               SET_INDENT (8);
               PRINT (" ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )");
               PRINT_LINE;
               PRINT ("return  ADA_SQL.");
               PRINT (STRING(TRACER.TARGET_TYPE.SCHEMA_UNIT.NAME.all) &
                     "_TYPE_PACKAGE.");
               PRINT (STRING(TRACER.TARGET_TYPE.NAME.all) & "_TYPE");
               PRINT_LINE;
               PRINT ("renames CONVERT_R;");
               PRINT_LINE;
               BLANK_LINE;
            end if;
            TRACER := TRACER.NEXT_TYPE;
         end loop;
         SET_INDENT (4);
         PRINT ("end ");
         PRINT (STRING(CURRENT_SCHEMA.NAME.all));
         PRINT (";");
         PRINT_LINE;
         BLANK_LINE;
      end loop;
      SET_INDENT (2);
      PRINT ("end CONVERT_TO;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
end POST_PROCESSING;
```

```
end CONVERT_TO;
```

3.11.46 package intos.ada

```
-- intos.ada -- post process data structures for INTO procedures

with DDL_DEFINITIONS;
package INTO is

-- The INTO procedures convert the internal representation of returned
-- database results to the program types required by the application.  Each
-- INTO procedure required is created by instantiating one of four generic
-- procedures.

-- Each of the four generic procedures is used with a specific class of types.
-- These classes are (the only classes currently supported by the application
-- scanner):
--
-- (1) integer and enumeration
-- (2) floating point
-- (3) unconstrained strings
-- (4) constrained strings

-- In what follows, type_name denotes the fully qualified name of a program
-- type.  If the program type is defined within the DDL, then type_name will
-- be of the form library_unit.ADA_SQL.type_simple_name.  If the program type
-- is a predefined one (i.e., in STANDARD or DATABASE), then type_name will be
-- of the form library_unit.type_simple_name.

-- Returned database strings are converted character by character to the
-- program type required by an INTO procedure.  (This conversion is somewhat
-- redundant if the components of the program type are CHARACTERs; see
-- discussion in chartos.ada.)  For each type used as a component of a string
-- type to be returned by an INTO procedure, a function must be written to
-- convert from type CHARACTER (internal representation) to that type.  This
-- (overloaded) function is called CONVERT_CHARACTER_TO_COMPONENT, and is
-- further described in chartos.ada.  It is used by the instantiations
-- generated here, but its name does not show up in the code because it is
-- passed to the generics as a default generic parameter.  INTO.REQUIRED_FOR
-- (see below) determines if an INTO procedure returning a string is being
-- specified, and, if so, calls CONVERT_CHARACTER_TO_COMPONENT.REQUIRED_FOR
-- the appropriate component type, to indicate that the component conversion
-- function must also be generated.

-- In the presentation of the generated instantiations for INTO procedures
-- returning strings, component_type_name represents the fully qualified name
-- of the component type.  If this type is defined in the DDL, then
-- component_type_name will be of the form library_unit.ADA_SQL.component_-
-- type_simple_name.  If this type is predefined, then component_type_name
-- will be of the form library_unit.component_type_simple_name.  (In the case
```

```
-- of STANDARD, the hand-generated runtime example used just type_simple_name,
-- but we can generate STANDARD.type_simple_name without hurting anything, and
-- saving the trouble of coding to detect the special case.)

-- Instantiating the generic INTO procedure for a constrained string type
-- requires passing the index subtype as a generic actual parameter.  As
-- discussed in indexs.ada, this subtype is often anonymous, based on typical
-- declarations of string types.  In such cases, we generate a declaration of
-- an index subtype, also as described in indexs.ada.  INTO.REQUIRED_FOR (see
-- below) determines if such a subtype declaration must be generated for the
-- type it is processing, and calls INDEX_SUBTYPE.REQUIRED_FOR any string
-- types requiring index subtypes to be generated.

-- If the index subtype that would otherwise be generated would have the same
-- bounds as the subtype used to declare the string type, then the latter
-- subtype is used for the index subtype and a new index subtype declaration
-- is not generated.  In the following example, NAME_INDEX would be used as
-- the index subtype for array NAME; a new index subtype declaration would not
-- be generated:
--
--   type NAME_INDEX is range 1 .. 20;
--   type NAME is array ( NAME_INDEX range 1 .. 20 ) of CHARACTER;

-- In the presentation of the generated instantiations for INTO procedures
-- returning constrained strings, index_subtype_name represents the
-- appropriately qualified name of the index subtype.  If this subtype must be
-- generated, then index_subtype_name will be of the form ADA_SQL.library_-
-- unit_INDEX_PACKAGE.type_simple_name_INDEX, where library_unit is the name
-- of the library unit in which the string type to be returned is declared,
-- and type_simple_name is the simple name of the type to be returned.  If a
-- suitable index subtype is defined in the user-written DDL, then index_-
-- subtype_name will be of the form library_unit.ADA_SQL.index_subtype_-
-- simple_name, where library_unit is the name of the DDL library unit
-- declaring the index subtype.  If a suitable index subtype is predefined
-- (e.g., in DATABASE), then index_subtype_name will be of the form library_-
-- unit.index_subtype_simple_name, where library_unit is the name of the
-- predefined library unit.

-- In the presentation of the generated instantiations for INTO procedures
-- returning unconstrained strings, index_subtype_name represents the fully
-- qualified name of the index subtype.  If this subtype is defined in the
-- DDL, then index_subtype_name will be of the form library_unit.ADA_SQL.-
-- index_subtype_simple_name.  If this subtype is predefined, then index_-
-- subtype_name will be of the form library_unit.index_subtype_simple_name.

-- The INTO procedure for an integer or enumeration type is created with the
-- following instantiation:
--
--   procedure INTO is new
```

```
--    ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INTO ( type_name );

-- The INTO procedure for a floating point type is created with the following
-- instantiation:
--
--   procedure INTO is new ADA_SQL_FUNCTIONS.FLOAT_INTO ( type_name );

-- The INTO procedure for an unconstrained string type is created with the
-- following instantiation:
--
--   procedure INTO is new
--    ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_INTO
--    ( index_subtype_name , component_type_name , type_name );

-- The INTO procedure for a constrained string type is created with the
-- following instantiation:
--
--   procedure INTO is new
--    ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INTO
--    ( index_subtype_name , component_type_name , type_name );

-- The information required to generate the INTO procedure appropriate for a
-- type is:
--
-- (1) Fully qualified name of that type
--
-- (2) Simple name of the type (some elements of information are redundant,
--     but are listed here as they wind up in different sections of the
--     generated code)
--
-- (3) Name of the library unit in which the type is declared
--
-- (4) Class of the type (integer, enumeration, floating point, unconstrained
--     string, constrained string)
--
-- (5) For strings, identity of the subtype used to declare the array index
--
--     (a) Fully qualified name of the subtype
--
--     (b) For constrained strings, bounds of the subtype
--
-- (6) For constrained strings, bounds of the array index
--
-- (7) For strings, fully qualified name of the component type

-- All this information can be deduced from the ACCESS_FULL_NAME_DESCRIPTOR
-- for a type.  INTO.REQUIRED_FOR flags that an INTO procedure is to be
-- instantiated for the type specified by its ACCESS_FULL_NAME_DESCRIPTOR.  It
-- is called whenever the necessity for an INTO procedure is found; it ignores
```

```
-- duplicate requests.

  procedure REQUIRED_FOR
            ( PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- INTO.POST_PROCESSING produces the code instantiating the INTO procedures:

  procedure POST_PROCESSING;

end INTO;
```

3.11.47 package intob.ada

```
-- intob.ada -- post process data structures for INTO procedures

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, INDEX_SUBTYPE,
     CONVERT_CHARACTER_TO_COMPONENT, DATABASE;
use  TEXT_PRINT;
package body INTO is

   use DDL_DEFINITIONS;
   use DATABASE;

   type REQUIRED_FOR_ENTRY_RECORD;
   type REQUIRED_FOR_ENTRY is access REQUIRED_FOR_ENTRY_RECORD;

   type REQUIRED_FOR_ENTRY_RECORD is
      record
         FULL_NAME_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                   DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         NEXT_REQUIRED_FOR    : REQUIRED_FOR_ENTRY;
      end record;

   REQUIRED_FOR_LIST : REQUIRED_FOR_ENTRY := new REQUIRED_FOR_ENTRY_RECORD;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.FULL_PACKAGE_NAME.all > RIGHT.FULL_PACKAGE_NAME.all then
      return TRUE;
   elsif LEFT.FULL_PACKAGE_NAME.all /= RIGHT.FULL_PACKAGE_NAME.all then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";
```

```
function INDEX_SUBTYPE_REQUIRED
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_LO /=
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_MIN or else
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_HI /=
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_MAX then
      return TRUE;
   else
      return FALSE;
   end if;
end INDEX_SUBTYPE_REQUIRED;

procedure REQUIRED_FOR
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST;
   -- Order list by fully-qualified component type name.
begin
   while TRACER.NEXT_REQUIRED_FOR /= null and then
      PROGRAM_TYPE >= TRACER.NEXT_REQUIRED_FOR.FULL_NAME_DESCRIPTOR loop
      TRACER := TRACER.NEXT_REQUIRED_FOR;
   end loop;
   if PROGRAM_TYPE /= TRACER.FULL_NAME_DESCRIPTOR then
      TRACER.NEXT_REQUIRED_FOR :=
         new REQUIRED_FOR_ENTRY_RECORD'
            (FULL_NAME_DESCRIPTOR => PROGRAM_TYPE,
             NEXT_REQUIRED_FOR    => TRACER.NEXT_REQUIRED_FOR);
      if PROGRAM_TYPE.TYPE_IS.TY_PE = DDL_DEFINITIONS.STR_ING then
         CONVERT_CHARACTER_TO_COMPONENT.REQUIRED_FOR
         (PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE.FULL_NAME);
         if PROGRAM_TYPE.TYPE_IS.CONSTRAINED and then
            INDEX_SUBTYPE_REQUIRED (PROGRAM_TYPE) then
            INDEX_SUBTYPE.REQUIRED_FOR (PROGRAM_TYPE.TYPE_IS);
         end if;
      end if;
   end if;
end REQUIRED_FOR;

procedure POST_PROCESSING is
   TRACER : REQUIRED_FOR_ENTRY := REQUIRED_FOR_LIST.NEXT_REQUIRED_FOR;
begin
   while TRACER /= null loop
      SET_INDENT (2);
      PRINT ("procedure INTO is new ");
      PRINT_LINE;
      SET_INDENT (4);
      case TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.TY_PE is
         when DDL_DEFINITIONS.REC_ORD => null;  -- should never occur.
```

```
            when DDL_DEFINITIONS.ENUMERATION | DDL_DEFINITIONS.INT_EGER =>
               PRINT ("ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_INTO");
               PRINT_LINE;
               SET_INDENT (6);
               PRINT ("( ");
            when DDL_DEFINITIONS.FL_OAT =>
               PRINT ("ADA_SQL_FUNCTIONS.FLOAT_INTO");
               PRINT_LINE;
               SET_INDENT (6);
               PRINT ("( ");
            when DDL_DEFINITIONS.STR_ING =>
               if TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.CONSTRAINED then
                  PRINT ("ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_INTO");
               else
                  PRINT ("ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_INTO");
               end if;
               SET_INDENT (6);
               PRINT ("( ");
               if TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.CONSTRAINED and then
                  INDEX_SUBTYPE_REQUIRED (TRACER.FULL_NAME_DESCRIPTOR) then
                  PRINT ("ADA_SQL.");
                  PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.SCHEMA_UNIT.NAME.all)
                         & "_INDEX_PACKAGE.");
                  PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all) &
                         "_INDEX");
               else
                  PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.
                         INDEX_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all) & ".");
                  PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.
                         INDEX_TYPE.FULL_NAME.NAME.all));
               end if;
               PRINT (",");
               PRINT_LINE;
               SET_INDENT (8);
               PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.
                      ARRAY_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all) & ".");
               PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.TYPE_IS.
                      ARRAY_TYPE.FULL_NAME.NAME.all));
               PRINT (",");
               PRINT_LINE;
         end case;
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.FULL_NAME_DESCRIPTOR.NAME.all));
         PRINT (" );");
         PRINT_LINE;
         BLANK_LINE;
         TRACER := TRACER.NEXT_REQUIRED_FOR;
      end loop;
   end POST_PROCESSING;
```

```
end INTO;
```

3.11.48 package pgmconvs.ada

```
-- pgmconvs.ada - post process data strucs for L_CONVERT & R_CONVERT functions

with DDL_DEFINITIONS;
 use DDL_DEFINITIONS;
package PROGRAM_CONVERSION is

-- The various Ada/SQL subprograms that have parameters of program types
-- convert the values of those parameters to a standard internal type, SQL_-
-- OBJECT, for entry into their data structures.  To perform these
-- conversions, the subprograms call L_CONVERT to convert their left or only
-- parameter, and R_CONVERT to convert their right parameter (if any).  Each
-- L_CONVERT function is created by instantiating one of six generic
-- functions, and the R_CONVERT functions are simply renamed from the
-- corresponding L_CONVERT functions.  The application scanner produces both
-- L_CONVERT and R_CONVERT functions for each program type used as a parameter
-- to an Ada/SQL subprogram and requiring conversion to SQL_OBJECT.

-- Each of the six generic functions is used with a specific class of types.
-- These classes are (the only classes currently supported by the application
-- scanner):
--
-- (1) integer and enumeration
-- (2) floating point
-- (3) unconstrained strings with CHARACTER components
-- (4) unconstrained strings with components of a type derived from CHARACTER
-- (5) constrained strings with CHARACTER components
-- (6) constrained strings with components of a type derived from CHARACTER

-- In what follows, type_name denotes the fully qualified name of a program
-- type.  If the program type is defined within the DDL, then type_name will
-- be of the form library_unit.ADA_SQL.type_simple_name.  If the program type
-- is a predefined one (i.e., in STANDARD or DATABASE), then type_name will be
-- of the form library_unit.type_simple_name.

-- Two considerations are relevant to the conversion functions for strings:
--
-- (1) Whether the components of the string are CHARACTERs or are of a type
--     derived from CHARACTER
--
-- (2) For constrained strings, whether or not the index subtype is anonymous.

-- Strings whose components are of type CHARACTER can be converted directly to
-- their internal representation within an SQL_OBJECT using an Ada type
-- conversion on the entire string value.  This is because the internal
-- representation of a string is as a STRING, which is an array of CHARACTERs.
```

```
-- Strings whose components are not of type CHARACTER cannot be converted
-- using an Ada type conversion on the entire string, but must be converted
-- character by character.  For each type, other than CHARACTER, used as a
-- component of a string to be converted to internal representation, a
-- function must be written to convert from that type to type CHARACTER.  This
-- (overloaded) function is called CONVERT_COMPONENT_TO_CHARACTER, and is
-- further described in comptos.ada.  It is used by the instantiations
-- generated here, but its name does not show up in the code because it is
-- passed to the generics as a default generic parameter.  PROGRAM_-
-- CONVERSION.REQUIRED_FOR (see below) determines if the conversion of a
-- string with non-CHARACTER components is being specified, and, if so, calls
-- CONVERT_COMPONENT_TO_CHARACTER.REQUIRED_FOR the appropriate component type,
-- to indicate that the component conversion function must also be generated.

-- In the presentation of the generated instantiations for conversions of
-- strings with non-CHARACTER components, component_type_name represents the
-- fully qualified name of the component type.  If this type is defined in the
-- DDL, then component_type_name will be of the form library_unit.ADA_SQL.-
-- component_type_simple_name.  If this type is predefined, then component_-
-- type_name will be of the form library_unit.component_type_simple_name.

-- Instantiating the generic string conversion routine for a constrained
-- string type requires passing the index subtype as a generic actual
-- parameter.  As discussed in indexs.ada, this subtype is often anonymous,
-- based on typical declarations of string types.  In such cases, we generate
-- a declaration of an index subtype, also as described in indexs.ada.
-- PROGRAM_CONVERSION.REQUIRED_FOR (see below) determines if such a subtype
-- declaration must be generated for the conversion it is processing, and
-- calls INDEX_SUBTYPE.REQUIRED_FOR any string types requiring index subtypes
-- to be generated.

-- If the index subtype that would otherwise be generated would have the same
-- bounds as the subtype used to declare the string type, then the latter
-- subtype is used for the index subtype and a new index subtype declaration
-- is not generated.  In the following example, NAME_INDEX would be used as
-- the index subtype for array NAME; a new index subtype declaration would not
-- be generated:
--
--   type NAME_INDEX is range 1 .. 20;
--   type NAME is array ( NAME_INDEX range 1 .. 20 ) of CHARACTER;

-- In the presentation of the generated instantiations for conversions of
-- constrained strings, index_subtype_name represents the appropriately
-- qualified name of the index subtype.  If this subtype must be generated,
-- then index_subtype_name will be of the form ADA_SQL.library_unit_INDEX_-
-- PACKAGE.type_simple_name_INDEX, where library_unit is the name of the
-- library unit in which the string type to be converted is declared, and
-- type_simple_name is the simple name of the type to be converted.  If a
-- suitable index subtype is defined in the user-written DDL, then index_-
```

```
-- subtype_name will be of the form library_unit.ADA_SQL.index_subtype_-
-- simple_name, where library_unit is the name of the DDL library unit
-- declaring the index subtype.  If a suitable index subtype is predefined
-- (e.g., in DATABASE), then index_subtype_name will be of the form library_-
-- unit.index_subtype_simple_name, where library_unit is the name of the
-- predefined library unit.

-- In the presentation of the generated instantiations for conversions of
-- unconstrained strings, index_subtype_name represents the fully qualified
-- name of the index subtype.  If this subtype is defined in the DDL, then
-- index_subtype_name will be of the form library_unit.ADA_SQL.index_subtype_-
-- simple_name.  If this subtype is predefined, then index_subtype_name will
-- be of the form library_unit.index_subtype_simple_name.

-- The L_CONVERT function for an integer or enumeration type is created with
-- the following instantiation:
--
--   function L_CONVERT is new
--    ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT ( type_name );

-- The L_CONVERT function for a floating point type is created with the
-- following instantiation:
--
--   function L_CONVERT is new ADA_SQL_FUNCTIONS.FLOAT_CONVERT ( type_name );

-- The L_CONVERT function for an unconstrained string with CHARACTER
-- components is created with the following instantiation:
--
--   function L_CONVERT is new
--    ADA_SQL_FUNCTIONS.UNCONSTRAINED_CHARACTER_STRING_CONVERT
--    ( index_subtype_name , type_name );

-- The L_CONVERT function for an unconstrained string with components of a
-- type derived from CHARACTER is created with the following instantiation:
--
--   function L_CONVERT is new
--    ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_CONVERT
--    ( index_subtype_name , component_type_name , type_name );

-- The L_CONVERT function for a constrained string with CHARACTER components
-- is created with the following instantiation:
--
--   function L_CONVERT is new
--    ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT
--    ( index_subtype_name , type_name );

-- The L_CONVERT function for a constrained string with components of a type
-- derived from CHARACTER is created with the following instantiation:
--
```

```
--   function L_CONVERT is new
--    ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_CONVERT
--    ( index_subtype_name , component_type_name , type_name );

-- An R_CONVERT function is created for each L_CONVERT function by renaming:
--
--   function R_CONVERT ( R : type_name ) return ADA_SQL_FUNCTIONS.SQL_OBJECT
--    renames L_CONVERT;

-- The information required to generate the conversion function appropriate
-- for a type is:
--
-- (1) Fully qualified name of the type
--
-- (2) Simple name of the type (some elements of information are redundant,
--     but are listed here as they wind up in different sections of the
--     generated code)
--
-- (3) Name of the library unit in which the type is declared
--
-- (4) Class of the type (integer, enumeration, floating point, unconstrained
--     string with CHARACTER components, unconstrained string with components
--     derived from CHARACTER, constrained string with CHARACTER components,
--     constrained string with components derived from CHARACTER)
--
-- (5) For strings, identity of the subtype used to declare the array index
--
--     (a) Fully qualified name of the subtype
--
--     (b) For constrained strings, bounds of the subtype
--
-- (6) For constrained strings, bounds of the array index
--
-- (7) For strings, fully qualified name of the component type
--
-- (8) For strings, whether or not the component type is CHARACTER

-- All this information can be deduced from the ACCESS_FULL_NAME_DESCRIPTOR
-- for a type.  PROGRAM_CONVERSION.REQUIRED_FOR flags that a conversion
-- routine is to be instantiated for the type specified by its ACCESS_FULL_-
-- NAME_DESCRIPTOR.  It is called whenever the necessity for a conversion is
-- found; it ignores duplicate conversion requests.

  procedure REQUIRED_FOR
            ( PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- PROGRAM_CONVERSION.POST_PROCESSING produces the code instantiating the
-- L_CONVERT functions and renaming them as R_CONVERT:
```

```
  procedure POST_PROCESSING;

end PROGRAM_CONVERSION;
```

3.11.49 package pgmconvb.ada

```
-- pgmconvb.ada

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, DATABASE, INDEX_SUBTYPE,
   CONVERT_COMPONENT_TO_CHARACTER, PREDEFINED_TYPE;
use  TEXT_PRINT;
package body PROGRAM_CONVERSION is

   use DDL_DEFINITIONS, DATABASE;

   type CONVERSION_KIND is
      (INTEGER_AND_ENUMERATION,
       FLOAT,
       UNCONSTRAINED_CHARACTER_STRING,
       UNCONSTRAINED_STRING,
       CONSTRAINED_CHARACTER_STRING,
       CONSTRAINED_STRING);

   type CONVERSION_ENTRY_RECORD;
   type CONVERSION_ENTRY is access CONVERSION_ENTRY_RECORD;

   type CONVERSION_ENTRY_RECORD is
      record
         PROGRAM_TYPE    : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                 DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         KIND            : CONVERSION_KIND;
         NEXT_CONVERSION : CONVERSION_ENTRY;
      end record;

   CONVERSION_LIST : CONVERSION_ENTRY := new CONVERSION_ENTRY_RECORD;

function BASE_TYPE_IS_CHARACTER
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR)
   return BOOLEAN is
begin
   if PROGRAM_TYPE = PREDEFINED_TYPE.STANDARD.CHARACTER then
      return TRUE;
   else
      return FALSE;
   end if;
end BASE_TYPE_IS_CHARACTER;

function INDEX_SUBTYPE_REQUIRED
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
```

```
begin
   if PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_LO /=
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_MIN or else
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_HI /=
      PROGRAM_TYPE.TYPE_IS.ARRAY_RANGE_MAX then
      return TRUE;
   else
      return FALSE;
   end if;
end INDEX_SUBTYPE_REQUIRED;

function GET_KIND_FOR
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return CONVERSION_KIND is
begin
   case PROGRAM_TYPE.TYPE_IS.WHICH_TYPE is
      when DDL_DEFINITIONS.ENUMERATION | DDL_DEFINITIONS.INT_EGER =>
         return INTEGER_AND_ENUMERATION;
      when DDL_DEFINITIONS.FL_OAT =>
         return FLOAT;
      when DDL_DEFINITIONS.STR_ING =>
         if not PROGRAM_TYPE.TYPE_IS.CONSTRAINED then
            if BASE_TYPE_IS_CHARACTER (PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE) then
               return UNCONSTRAINED_CHARACTER_STRING;
            else
               CONVERT_COMPONENT_TO_CHARACTER.REQUIRED_FOR
                  (PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE.FULL_NAME);
               return UNCONSTRAINED_STRING;
            end if;
         else
            if INDEX_SUBTYPE_REQUIRED (PROGRAM_TYPE) then
               INDEX_SUBTYPE.REQUIRED_FOR (PROGRAM_TYPE.TYPE_IS);
            end if;
            if BASE_TYPE_IS_CHARACTER (PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE) then
               return CONSTRAINED_CHARACTER_STRING;
            else
               CONVERT_COMPONENT_TO_CHARACTER.REQUIRED_FOR
                  (PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE.FULL_NAME);
               return CONSTRAINED_STRING;
            end if;
         end if;
      when others =>
         raise PROGRAM_ERROR; -- should never occur.
   end case;
end GET_KIND_FOR;

function ">="
   (LEFT , RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
```

```
begin
   if LEFT.FULL_PACKAGE_NAME.all > RIGHT.FULL_PACKAGE_NAME.all then
      return TRUE;
   elsif LEFT.FULL_PACKAGE_NAME.all /= RIGHT.FULL_PACKAGE_NAME.all then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";

procedure REQUIRED_FOR
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   TRACER : CONVERSION_ENTRY := CONVERSION_LIST;
begin
   while TRACER.NEXT_CONVERSION /= null and then
      PROGRAM_TYPE >= TRACER.NEXT_CONVERSION.PROGRAM_TYPE loop
      TRACER := TRACER.NEXT_CONVERSION;
   end loop;
   if PROGRAM_TYPE /= TRACER.PROGRAM_TYPE then
      TRACER.NEXT_CONVERSION := new CONVERSION_ENTRY_RECORD'
                                   (PROGRAM_TYPE => PROGRAM_TYPE,
                                    KIND         => GET_KIND_FOR (PROGRAM_TYPE),
                                    NEXT_CONVERSION => TRACER.NEXT_CONVERSION);
   end if;
end REQUIRED_FOR;

procedure POST_PROCESSING is
   TRACER : CONVERSION_ENTRY := CONVERSION_LIST.NEXT_CONVERSION;

   procedure PRINT_FULLY_QUALIFIED_NAME
      (PROGRAM_TYPE : in DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   begin
      PRINT (STRING(PROGRAM_TYPE.FULL_PACKAGE_NAME.all) & ".");
      PRINT (STRING(PROGRAM_TYPE.NAME.all));
   end PRINT_FULLY_QUALIFIED_NAME;

   procedure PRINT_GENERATED_INDEX_SUBTYPE_NAME
      (PROGRAM_TYPE : in DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   begin
      PRINT ("ADA_SQL.");
      PRINT (STRING(PROGRAM_TYPE.TYPE_IS.FULL_NAME.SCHEMA_UNIT.NAME.all) &
             "_INDEX_PACKAGE.");
      PRINT (STRING(PROGRAM_TYPE.TYPE_IS.FULL_NAME.NAME.all) & "_INDEX ");
   end PRINT_GENERATED_INDEX_SUBTYPE_NAME;

begin
   while TRACER /= null loop
```

```
SET_INDENT (2);
PRINT ("function L_CONVERT is new ");
PRINT_LINE;
SET_INDENT (4);
case TRACER.KIND is
   when INTEGER_AND_ENUMERATION =>
      PRINT ("ADA_SQL_FUNCTIONS.INTEGER_AND_ENUMERATION_CONVERT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
   when FLOAT =>
      PRINT ("ADA_SQL_FUNCTIONS.FLOAT_CONVERT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
   when UNCONSTRAINED_CHARACTER_STRING =>
      PRINT ("ADA_SQL_FUNCTIONS.UNCONSTRAINED_CHARACTER_STRING_CONVERT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_FULLY_QUALIFIED_NAME
         (TRACER.PROGRAM_TYPE.TYPE_IS.INDEX_TYPE.FULL_NAME);
      PRINT (",");
      PRINT_LINE;
      SET_INDENT (8);
      PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
   when UNCONSTRAINED_STRING =>
      PRINT ("ADA_SQL_FUNCTIONS.UNCONSTRAINED_STRING_CONVERT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_FULLY_QUALIFIED_NAME
         (TRACER.PROGRAM_TYPE.TYPE_IS.INDEX_TYPE.FULL_NAME);
      PRINT (",");
      PRINT_LINE;
      SET_INDENT (8);
      PRINT_FULLY_QUALIFIED_NAME
         (TRACER.PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE.FULL_NAME);
      PRINT (",");
      PRINT_LINE;
      PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
   when CONSTRAINED_CHARACTER_STRING =>
      PRINT ("ADA_SQL_FUNCTIONS.CONSTRAINED_CHARACTER_STRING_CONVERT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      if INDEX_SUBTYPE_REQUIRED (TRACER.PROGRAM_TYPE) then
```

```
            PRINT_GENERATED_INDEX_SUBTYPE_NAME (TRACER.PROGRAM_TYPE);
         else
            PRINT_FULLY_QUALIFIED_NAME
               (TRACER.PROGRAM_TYPE.TYPE_IS.INDEX_TYPE.FULL_NAME);
         end if;
         PRINT (",");
         PRINT_LINE;
         SET_INDENT (8);
         PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
      when CONSTRAINED_STRING =>
         PRINT ("ADA_SQL_FUNCTIONS.CONSTRAINED_STRING_CONVERT");
         PRINT_LINE;
         SET_INDENT (6);
         PRINT ("( ");
         if INDEX_SUBTYPE_REQUIRED (TRACER.PROGRAM_TYPE) then
            PRINT_GENERATED_INDEX_SUBTYPE_NAME (TRACER.PROGRAM_TYPE);
         else
            PRINT_FULLY_QUALIFIED_NAME
               (TRACER.PROGRAM_TYPE.TYPE_IS.INDEX_TYPE.FULL_NAME);
         end if;
         PRINT (",");
         PRINT_LINE;
         SET_INDENT (8);
         PRINT_FULLY_QUALIFIED_NAME
            (TRACER.PROGRAM_TYPE.TYPE_IS.ARRAY_TYPE.FULL_NAME);
         PRINT (",");
         PRINT_LINE;
         PRINT_FULLY_QUALIFIED_NAME (TRACER.PROGRAM_TYPE);
   end case;
   PRINT (" );");
   PRINT_LINE;
   BLANK_LINE;
   SET_INDENT (2);
   PRINT ("function R_CONVERT");
   PRINT_LINE;
   SET_INDENT (4);
   PRINT ("( R : ");
   PRINT (STRING(TRACER.PROGRAM_TYPE.FULL_PACKAGE_NAME.all) & ".");
   PRINT (STRING(TRACER.PROGRAM_TYPE.NAME.all));
   PRINT (" )");
   PRINT_LINE;
   PRINT ("return ADA_SQL_FUNCTIONS.SQL_OBJECT");
   PRINT_LINE;
   PRINT ("renames L_CONVERT;");
   PRINT_LINE;
   BLANK_LINE;
   TRACER := TRACER.NEXT_CONVERSION;
 end loop;
end POST_PROCESSING;
```

```
end PROGRAM_CONVERSION;
```

3.11.50 package predefs.ada

```
-- predefs.ada - post process data structure for optional predefined text

package PREDEFINED is

-- Certain Ada/SQL constructs require that predefined (non-parameterized) text
-- be generated for them.  These constructs are described below, along with
-- enumeration values used to refer to them and the text that must be
-- generated.

-- STAR_TYPE_DECLARATION
  -- type STAR_TYPE is ( '*' );
  --
  -- The enumeration value '*' must be visible to the user program if it is
  -- referenced as COUNT ( '*' ) or any flavor of SELEC ( '*' ... .
  -- Regrettably, this is not the kind of enumeration literal than can be made
  -- visible by renaming, so we have to generate our own type declaration.

-- UNTYPED_COUNT_STAR_FUNCTION
  -- Code generated is in two parts: (1) a specification, and (2) body parts.
  -- (Looking back on this, I don't know why I didn't just set up a generic
  -- function to do the whole ball of wax, instead of having to do a body
  -- here!  Maybe we'll change it later!)
  --
  -- Specification:
  --
  -- function COUNT ( STAR : STAR_TYPE ) return ADA_SQL_FUNCTIONS.SQL_OBJECT;
  --
  -- Body parts:
  --
  -- function COUNT_FUNCTION is new
  --  ADA_SQL_FUNCTIONS.COUNT_STAR ( ADA_SQL_FUNCTION.SQL_OBJECT );
  --
  -- function COUNT ( STAR : STAR_TYPE )
  --  return ADA_SQL_FUNCTIONS.SQL_OBJECT is
  -- begin
  --   return COUNT_FUNCTION;
  -- end COUNT;
  --
  -- Must be generated if COUNT ( '*' ) is used in a context where its result
  -- will not be strongly typed.

-- TYPED_COUNT_STAR_FUNCTION
  -- Code generated is in two parts: (1) a specification, and (2) body parts.
  -- See comment for UNTYPED_COUNT_STAR_FUNCTION, above.
  --
  -- Specification:
```

```
    --
    -- function COUNT ( STAR : STAR_TYPE )
    --  return ADA_SQL.DATABASE_TYPE_PACKAGE.INT_TYPE;
    --
    -- Body parts:
    --
    -- function COUNT_FUNCTION is new
    --  ADA_SQL_FUNCTIONS.COUNT_STAR ( ADA_SQL.DATABASE_TYPE_PACKAGE.INT_TYPE );
    --
    -- function COUNT ( STAR : STAR_TYPE )
    --  return ADA_SQL.DATABASE_TYPE_PACKAGE.INT_TYPE is
    -- begin
    --   return COUNT_FUNCTION;
    -- end COUNT;
    --
    -- Must be generated if COUNT ( '*' ) is used in a context where its result
    -- will be strongly typed.  (Note that strongly typed COUNT ( '*' ) always
    -- returns a result of the database type corresponding to DATABASE.INT.)

  -- CLOSE_PROCEDURE
    -- procedure CLOSE ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
    --  renames ADA_SQL_FUNCTIONS.CLOSE;
    --
    -- Must be generated if CLOSE is called.

  -- DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY
    -- procedure DECLAR
    --          ( CURSOR     : in out ADA_SQL_FUNCTIONS.CURSOR_NAME;
    --            CURSOR_FOR : in     ADA_SQL_FUNCTIONS.SQL_OBJECT;
    --            ORDER_BY   : in     DATABASE.COLUMN_NUMBER )
    --  renames ADA_SQL_FUNCTIONS.DECLAR;
    --
    -- Must be generated if DECLAR is called with a single ORDER BY column,
    -- specified as a column number.

  -- DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY
    -- procedure DECLAR
    --          ( CURSOR     : in out ADA_SQL_FUNCTIONS.CURSOR_NAME;
    --            CURSOR_FOR : in     ADA_SQL_FUNCTIONS.SQL_OBJECT;
    --            ORDER_BY   : in     ADA_SQL_FUNCTIONS.SQL_OBJECT :=
    --                                 ADA_SQL_FUNCTIONS.NULL_OBJECT )
    --  renames ADA_SQL_FUNCTIONS.DECLAR;
    --
    -- Must be generated if DECLAR is called, under conditions opposite to
    -- previous DECLAR.

  -- DELETE_SEARCHED_PROCEDURE
    -- procedure DELETE_FROM
    --          ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
```

```
  --              WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                            ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
  --  renames ADA_SQL_FUNCTIONS.DELETE_FROM;
  --
  -- Must be generated if the searched version of DELETE_FROM is called.  (The
  -- positioned version of DELETE_FROM is not supported in this version.)

-- FETCH_PROCEDURE
  -- procedure FETCH ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
  --  renames ADA_SQL_FUNCTIONS.FETCH;
  --
  -- Must be generated if FETCH is called.

-- INSERT_INTO_PROCEDURE
  -- procedure INSERT_INTO
  --            ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
  --              WHAT  : in ADA_SQL_FUNCTIONS.INSERT_ITEM )
  --  renames ADA_SQL_FUNCTIONS.INSERT_INTO;
  --
  -- Must be generated if INSERT_INTO is called.

-- VALUES_FUNCTION
  -- function VALUES return ADA_SQL_FUNCTIONS.INSERT_ITEM
  --  renames ADA_SQL_FUNCTIONS.VALUES;
  --
  -- Must be generated if the VALUES form of INSERT_INTO is used.

-- OPEN_PROCEDURE
  -- procedure OPEN ( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )
  --  renames ADA_SQL_FUNCTIONS.OPEN;
  --
  -- Must be generated if OPEN is called.

-- UPDATE_SEARCHED_PROCEDURE
  -- procedure UPDATE
  --            ( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;
  --              SET   : in ADA_SQL_FUNCTIONS.SQL_OBJECT;
  --              WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                            ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
  --  renames ADA_SQL_FUNCTIONS.UPDATE;
  --
  -- Must be generated if the searched version of UPDATE is called.  (The
  -- positioned version of UPDATE is not supported in this version.)

-- A value of PREDEFINED.TEXT_TYPE identifies a particular piece of predefined
-- text:

  type TEXT_TYPE is
      ( STAR_TYPE_DECLARATION,
```

```
          UNTYPED_COUNT_STAR_FUNCTION,
          TYPED_COUNT_STAR_FUNCTION,
          CLOSE_PROCEDURE,
          DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY,
          DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY,
          DELETE_SEARCHED_PROCEDURE,
          FETCH_PROCEDURE,
          INSERT_INTO_PROCEDURE,
          VALUES_FUNCTION,
          OPEN_PROCEDURE,
          UPDATE_SEARCHED_PROCEDURE );

-- PREDEFINED.TEXT_REQUIRED_FOR is called, for the appropriate predefined text
-- type, whenever it is determined that a piece of predefined text must be
-- generated.  (Duplicate calls for the same particular piece of predefined
-- text are fine; PREDEFINED.TEXT_REQUIRED_FOR automatically ignores duplicate
-- calls and only produces the required text once.)

   procedure TEXT_REQUIRED_FOR ( TEXT_OF : TEXT_TYPE );

-- Post processing for predefined text is done in two steps: (1) all required
-- predefined text is generated, in the order discussed above, except for the
-- body parts of the COUNT ( '*' ) functions, and (2) the body parts of the
-- COUNT ( '*' ) functions are generated.  These functions are handled by
-- PREDEFINED.TEXT_POST_PROCESSING_1 and PREDEFINED.TEXT_POST_PROCESSING_2.

   procedure TEXT_POST_PROCESSING_1;

   procedure TEXT_POST_PROCESSING_2;

end PREDEFINED;
```

3.11.51 package predefb.ada

```
-- predefb.ada - post process data structure for optional predefined text

with TEXT_PRINT, DDL_DEFINITIONS, DATABASE_TYPE, PREDEFINED_TYPE;
use  TEXT_PRINT;
package body PREDEFINED is

   NEED_STAR_TYPE_DECLARATION                   : BOOLEAN := FALSE;
   NEED_UNTYPED_COUNT_STAR_FUNCTION             : BOOLEAN := FALSE;
   NEED_TYPED_COUNT_STAR_FUNCTION               : BOOLEAN := FALSE;
   NEED_CLOSE_PROCEDURE                         : BOOLEAN := FALSE;
   NEED_DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY  : BOOLEAN := FALSE;
   NEED_DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY : BOOLEAN := FALSE;
   NEED_DELETE_SEARCHED_PROCEDURE               : BOOLEAN := FALSE;
   NEED_FETCH_PROCEDURE                         : BOOLEAN := FALSE;
   NEED_INSERT_INTO_PROCEDURE                   : BOOLEAN := FALSE;
   NEED_VALUES_FUNCTION                         : BOOLEAN := FALSE;
```

```
   NEED_OPEN_PROCEDURE                                  : BOOLEAN := FALSE;
   NEED_UPDATE_SEARCHED_PROCEDURE                       : BOOLEAN := FALSE;

procedure TEXT_REQUIRED_FOR
   (TEXT_OF : TEXT_TYPE) is
begin
   case TEXT_OF is
      when STAR_TYPE_DECLARATION =>
         NEED_STAR_TYPE_DECLARATION := TRUE;
      when UNTYPED_COUNT_STAR_FUNCTION =>
         NEED_UNTYPED_COUNT_STAR_FUNCTION := TRUE;
         NEED_STAR_TYPE_DECLARATION := TRUE;
      when TYPED_COUNT_STAR_FUNCTION =>
         NEED_TYPED_COUNT_STAR_FUNCTION := TRUE;
         NEED_STAR_TYPE_DECLARATION := TRUE;
         DATABASE_TYPE.REQUIRED_FOR (PREDEFINED_TYPE.DATABASE.INT.FULL_NAME);
      when CLOSE_PROCEDURE =>
         NEED_CLOSE_PROCEDURE := TRUE;
      when DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY =>
         NEED_DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY := TRUE;
      when DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY =>
         NEED_DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY := TRUE;
      when DELETE_SEARCHED_PROCEDURE =>
         NEED_DELETE_SEARCHED_PROCEDURE := TRUE;
      when FETCH_PROCEDURE =>
         NEED_FETCH_PROCEDURE := TRUE;
      when INSERT_INTO_PROCEDURE =>
         NEED_INSERT_INTO_PROCEDURE := TRUE;
      when VALUES_FUNCTION =>
         NEED_VALUES_FUNCTION := TRUE;
      when OPEN_PROCEDURE =>
         NEED_OPEN_PROCEDURE := TRUE;
      when UPDATE_SEARCHED_PROCEDURE =>
         NEED_UPDATE_SEARCHED_PROCEDURE := TRUE;
   end case;
end TEXT_REQUIRED_FOR;

procedure PRINT_DATABASE_INT is
   TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                 PREDEFINED_TYPE.DATABASE.INT;
begin
   PRINT ("ADA_SQL.");
   PRINT (STRING(TYPE_DES.FULL_NAME.SCHEMA_UNIT.NAME.all) & "_TYPE_PACKAGE.");
   PRINT (STRING(TYPE_DES.FULL_NAME.NAME.all) & "_TYPE");
end PRINT_DATABASE_INT;

procedure TEXT_POST_PROCESSING_1 is
begin
   if NEED_STAR_TYPE_DECLARATION then
```

```
      SET_INDENT (2);
      PRINT ("type STAR_TYPE is ('*');");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_UNTYPED_COUNT_STAR_FUNCTION then
      SET_INDENT (2);
      PRINT ("function COUNT");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( STAR : STAR_TYPE )");
      PRINT_LINE;
      PRINT ("return ADA_SQL_FUNCTIONS.SQL_OBJECT;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_TYPED_COUNT_STAR_FUNCTION then
      SET_INDENT (2);
      PRINT ("function COUNT");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( STAR : STAR_TYPE )");
      PRINT_LINE;
      PRINT ("return ");
      PRINT_DATABASE_INT;
      PRINT (";");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_CLOSE_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure CLOSE");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )");
      PRINT_LINE;
      PRINT ("renames ADA_SQL_FUNCTIONS.CLOSE;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY then
      SET_INDENT (2);
      PRINT ("procedure DECLAR");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( CURSOR     : in out ADA_SQL_FUNCTIONS.CURSOR_NAME;");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("CURSOR_FOR : in     ADA_SQL_FUNCTIONS.SQL_OBJECT;");
```

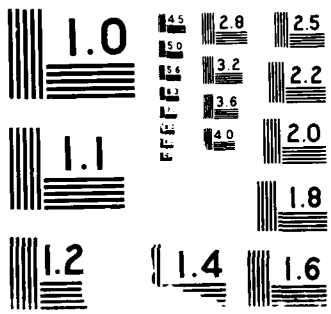
```
      PRINT_LINE;
      PRINT ("ORDER_BY    : in      DATABASE.COLUMN_NUMBER )");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("renames ADA_SQL_FUNCTIONS.DECLAR;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY then
      SET_INDENT (2);
      PRINT ("procedure DECLAR");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( CURSOR     : in out ADA_SQL_FUNCTIONS.CURSOR_NAME;");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("CURSOR_FOR : in      ADA_SQL_FUNCTIONS.SQL_OBJECT;");
      PRINT_LINE;
      PRINT ("ORDER_BY    : in      ADA_SQL_FUNCTIONS.SQL_OBJECT := ");
      PRINT_LINE;
      PRINT ("                       ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("renames ADA_SQL_FUNCTIONS.DECLAR;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_DELETE_SEARCHED_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure DELETE_FROM");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT := ");
      PRINT_LINE;
      PRINT ("            ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("renames ADA_SQL_FUNCTIONS.DELETE_FROM;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_FETCH_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure FETCH");
      PRINT_LINE;
      SET_INDENT (4);
```

```
      PRINT ("( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )");
      PRINT_LINE;
      PRINT ("renames ADA_SQL_FUNCTIONS.FETCH;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_INSERT_INTO_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure INSERT_INTO");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("WHAT  : in ADA_SQL_FUNCTIONS.INSERT_ITEM)");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("renames ADA_SQL_FUNCTIONS.INSERT_INTO;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_VALUES_FUNCTION then
      SET_INDENT (2);
      PRINT ("function VALUES");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("return  ADA_SQL_FUNCTIONS.INSERT_ITEM");
      PRINT_LINE;
      PRINT ("renames ADA_SQL_FUNCTIONS.VALUES;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_OPEN_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure OPEN");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( CURSOR : in out ADA_SQL_FUNCTIONS.CURSOR_NAME )");
      PRINT_LINE;
      PRINT ("renames ADA_SQL_FUNCTIONS.OPEN;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_UPDATE_SEARCHED_PROCEDURE then
      SET_INDENT (2);
      PRINT ("procedure UPDATE");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( TABLE : in ADA_SQL_FUNCTIONS.TABLE_NAME;");
```

```
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("SET   : in ADA_SQL_FUNCTIONS.SQL_OBJECT;");
      PRINT_LINE;
      PRINT ("WHERE : in ADA_SQL_FUNCTIONS.SQL_OBJECT := ");
      PRINT_LINE;
      PRINT ("            ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("renames ADA_SQL_FUNCTIONS.UPDATE;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
end TEXT_POST_PROCESSING_1;

procedure TEXT_POST_PROCESSING_2 is
begin
   if NEED_UNTYPED_COUNT_STAR_FUNCTION then
      SET_INDENT (2);
      PRINT ("function COUNT_FUNCTION is new ");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("ADA_SQL_FUNCTIONS.COUNT_STAR");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ADA_SQL_FUNCTIONS.SQL_OBJECT );");
      PRINT_LINE;
      BLANK_LINE;
      SET_INDENT (2);
      PRINT ("function COUNT");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("( STAR : STAR_TYPE )");
      PRINT_LINE;
      PRINT ("return ADA_SQL_FUNCTIONS.SQL_OBJECT is");
      PRINT_LINE;
      SET_INDENT (2);
      PRINT ("begin");
      PRINT_LINE;
      PRINT ("  return COUNT_FUNCTION;");
      PRINT_LINE;
      PRINT ("end COUNT;");
      PRINT_LINE;
      BLANK_LINE;
   end if;
   if NEED_TYPED_COUNT_STAR_FUNCTION then
      SET_INDENT (2);
      PRINT ("function COUNT_FUNCTION is new ");
      PRINT_LINE;
```

AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317 3/6

UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

```
        SET_INDENT (4);
        PRINT ("ADA_SQL_FUNCTIONS.COUNT_STAR");
        PRINT_LINE;
        SET_INDENT (6);
        PRINT ("( ");
        PRINT_DATABASE_INT;
        PRINT (" );");
        PRINT_LINE;
        BLANK_LINE;
        SET_INDENT (2);
        PRINT ("function COUNT");
        PRINT_LINE;
        SET_INDENT (4);
        PRINT ("( STAR : STAR_TYPE )");
        PRINT_LINE;
        PRINT ("return ");
        PRINT_DATABASE_INT;
        PRINT (" is");
        PRINT_LINE;
        SET_INDENT (2);
        PRINT ("begin");
        PRINT_LINE;
        PRINT ("  return COUNT_FUNCTION;");
        PRINT_LINE;
        PRINT ("end COUNT;");
        PRINT_LINE;
        BLANK_LINE;
   end if;
end TEXT_POST_PROCESSING_2;

end PREDEFINED;
```

3.11.52 package froms.ada

```
-- froms.ada - internal data structures for from clauses

with CORRELATION, DDL_DEFINITIONS;
package FROM_CLAUSE is

-- The information about from clauses that we must process is conceptually
-- simple: A from clause is a list of table references.  Unfortunately, there
-- are two complicating factors:
--
-- (1) A table reference may be either an exposed table (table name used by
--     itself) or a table to be referenced through a correlation name
--     (correlation name used with table name in table reference)
--
-- (2) The scopes of from clauses can be nested, and the semantics of
--     expressions requires that we keep track of how they are nested within
--     an SQL statement.  In particular, at any point within an SQL statement,
```

```
--      we must know about the from clauses at successively outer levels of
--      nesting within the statement.  It is not necessary for us to know
--      about other from clauses within the statement but whose scope does not
--      include the current point.

-- To handle situation (2), information about from clauses seen are kept on a
-- stack.  When we enter a new scope, the from clause information for that
-- scope is pushed onto the stack as the top entry.  When our processing of a
-- statement leaves a scope, the from clause information for that scope is
-- popped off the stack and forgotten (processing never re-enters a scope that
-- has been left).  Data structure entries of type FROM_CLAUSE.INFORMATION are
-- linked together to form the stack.  Each entry represents a from clause at
-- a single scope, and points to the entry (if any) for the next outer scope.
-- The entry for the innermost scope being processed is on the top of the
-- stack, and it is a pointer to this entry that is used by the calling
-- routines.  (FROM_CLAUSE.INFORMATION is the only data structure that is
-- visible outside this package.  Details on the storage of the information
-- are private; the routines available to access that information are
-- described below.)

  type INFORMATION_RECORD is private;

  type INFORMATION is access INFORMATION_RECORD;

-- A from clause at a single scope consists of a list of tables referenced at
-- that scope.  Each entry is of the following form (see private part for
-- details):

  type TABLE_ENTRY is private;

  -- When about to process a from clause at a new scope, FROM_CLAUSE.AT_NEW_-
  -- SCOPE is called to create a new stack entry for the scope.
  -- Called with:
  --   FROM_CLAUSE.INFORMATION for the scope just outer to the one about to
  --    be entered - this is the value returned by the last call to FROM_-
  --    CLAUSE.AT_NEW_SCOPE if the last scope-related action was to enter a
  --    new scope, or the value returned by the last call to FROM_CLAUSE.AT_-
  --    OUTER_SCOPE if the last scope-related action was to exit from a
  --    scope), or NULL if we are processing the first from clause in the
  --    statement (no outer scope)
  -- Returns:
  --   Pointer to stack entry just created for the new from clause.  This is
  --    the FROM_CLAUSE.INFORMATION value that will be passed to the other
  --    routines described below as the remainder of the statement at this
  --    scope is processed.  When processing of this scope is complete, it is
  --    the value that will be passed to FROM_CLAUSE.AT_OUTER_SCOPE to resume
  --    processing at the next outer scope (if any).

  function AT_NEW_SCOPE ( SCOPE : INFORMATION ) return INFORMATION;
```

```
-- When exiting from a scope, FROM_CLAUSE.AT_OUTER_SCOPE is called to pop
-- the information on the current scope off the stack, and return the
-- pointer to the from clause information for the next outer scope.  NULL
-- is returned if the information for the outermost scope of the current
-- statement is popped.

function AT_OUTER_SCOPE ( SCOPE : INFORMATION ) return INFORMATION;

-- As a from clause is processed, a list of tables named in that from clause
-- (either exposed or correlated) is created.  FROM_CLAUSE.NAMES_EXPOSED_-
-- TABLE is called to add an exposed table to the list, and FROM_CLAUSE.-
-- NAMES_CORRELATED_TABLE is called to add a correlated table to the list.

-- An exposed table is represented in the from clause information by its
-- ACCESS_TYPE_DESCRIPTOR, which is used as a parameter to FROM_CLAUSE.-
-- NAMES_EXPOSED_TABLE.  Before calling FROM_CLAUSE.NAMES_EXPOSED_TABLE, the
-- calling routine verifies that the table exists (thereby obtaining its
-- ACCESS_TYPE_DESCRIPTOR) and that its name is not already exposed in the
-- from clause being processed (by calling FROM_CLAUSE.EXPOSES_NAME for the
-- current scope).  FROM_CLAUSE.NAMES_EXPOSED_TABLE is
-- Called with:
--   The FROM_CLAUSE.INFORMATION pointer for the current scope
--   The ACCESS_TYPE_DESCRIPTOR pointer for the table named in the from
--    clause

procedure NAMES_EXPOSED_TABLE
          ( SCOPE : INFORMATION;
            TABLE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR );

-- A correlated table is represented in the from clause information by its
-- CORRELATION.NAME_DECLARED_ENTRY (see corrs.ada), which is used as a
-- parameter to FROM_CLAUSE.NAMES_CORRELATED_TABLE.  Before calling FROM_-
-- CLAUSE.NAMES_CORRELATED_TABLE, the calling routine verifies that the
-- correlation reference is valid (by calling CORRELATION.NAME_RETURNS_-
-- TABLE_LIST or CORRELATION.NAME_RETURNS_TABLE_NAME, and thereby obtaining
-- the pointer to the appropriate CORRELATION.NAME_DECLARED_ENTRY) and that
-- the correlation name is not already exposed in the from clause being
-- processed (by calling FROM_CLAUSE.EXPOSES_NAME for the current scope).
-- FROM_CLAUSE.NAMES_CORRELATED_TABLE is
-- Called with:
--   The FROM_CLAUSE.INFORMATION pointer for the current scope
--   The CORRELATION.NAME_DECLARED_ENTRY for the correlation name used in
--    the from clause

procedure NAMES_CORRELATED_TABLE
          ( SCOPE            : INFORMATION;
            CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY );

-- The remaining visible routines are used to interrogate the from clause
```

```
-- information:

-- FROM_CLAUSE.EXPOSES_NAME determines whether the given name has been
-- exposed within a from clause as either an exposed table name or a
-- correlation name.  If THIS_SCOPE_ONLY is TRUE, then it only checks the
-- from clause for the current scope.  This is used (1) when processing the
-- from clause to verify that no name is exposed more than once, and (2)
-- when processing a column specification containing a qualifier, in
-- contexts where the column specified must appear in a table named in the
-- from clause at the current scope (e.g., a grouping column).  If THIS_-
-- SCOPE_ONLY is FALSE, then FROM_CLAUSE.EXPOSES_NAME looks at successively
-- outer nested scopes, beginning with the innermost one, until it either
-- finds the given name or has checked the outermost scope.  This is used
-- when processing a column specification containing a qualifier, in
-- contexts where outer references are permitted (and hence the qualifier
-- may refer to an outer scope).  Specific parameters are:
--   (in)  The string representation of the name in question
--   (in)  Pointer for the current from clause scope
--   (in)  Flag to restrict search to current scope only
--   (out) If NULL, the given name is not included as an exposed table in
--         the from clause(s) searched.  Otherwise, the ACCESS_TYPE_-
--         DESCRIPTOR for the named table that is exposed in the from
--         clause.
--   (out) If NULL, the given name is not used as a correlation name in the
--         from clause(s) searched.  Otherwise, the CORRELATION.NAME_-
--         DECLARED_ENTRY for the named correlation name, which is used in
--         the from clause.  (Since we build from clause information in
--         accordance with SQL semantics, rejecting invalid constructs, the
--         last two parameters cannot both return non-NULL values from the
--         same call.)

procedure EXPOSES_NAME
          ( NAME              : in  STRING;
            SCOPE             : in  INFORMATION;
            THIS_SCOPE_ONLY   : in  BOOLEAN;
            EXPOSED_TABLE     : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            CORRELATION_NAME  : out CORRELATION.NAME_DECLARED_ENTRY );

-- FROM_CLAUSE.MAKES_COLUMN_VISIBLE determines whether the named column
-- appears in any of the tables listed in a from clause.  If THIS_SCOPE_ONLY
-- is true, then it only checks the from clause for the current scope.  This
-- is used when processing unqualified column specifications in contexts
-- where the column must be in a table at the current scope (e.g., grouping
-- column).  If THIS_SCOPE_ONLY is FALSE, then FROM_CLAUSE.MAKES_COLUMN_-
-- VISIBLE looks at successively outer nested scopes, beginning with the
-- innermost one, until it either finds a from clause referencing a table
-- containing the column, or has checked the outermost scope.  This is used
-- when processing unqualified column specifications in contexts where outer
-- references are permitted.  Specific parameters are:
```

```
--   (in)  String representation of the column name in question
--   (in)  Pointer for the current from clause scope
--   (in)  Flag to restrict search to current scope only
--   (out) TRUE if the first from clause found that names a table containing
--         the given column names more than one such table (this means that
--         the column specification is in error!), otherwise FALSE
--   (out) (valid only if the third parameter is FALSE) If NULL, then the
--         named column does not appear in any table named in the from
--         clause(s) searched.  Otherwise, the ACCESS_FULL_NAME_DESCRIPTOR
--         for the column (which contains information about which table the
--         column is in, its type, etc.)

procedure MAKES_COLUMN_VISIBLE
          ( NAME :
             in STRING;
            SCOPE :
             in INFORMATION;
            THIS_SCOPE_ONLY :
             in BOOLEAN;
            COLUMN_APPEARS_IN_MORE_THAN_ONE_TABLE :
             out BOOLEAN;
            DESCRIPTOR :
             out DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- FROM_CLAUSE.TABLES_AT_CURRENT_SCOPE returns the TABLE_ENTRY corresponding
-- to the first table defined at the current scope, given by its parameter.
-- It is used for processing SELECT *, to determine what tables contribute
-- to the *.

function TABLES_AT_CURRENT_SCOPE ( SCOPE : INFORMATION ) return TABLE_ENTRY;

-- FROM_CLAUSE.NEXT_TABLE returns information about the next table named at
-- the current scope.  It is used in conjunction with FROM_CLAUSE.TABLES_-
-- AT_CURRENT_SCOPE.  Specific parameters are:
--   (in out) On call: The TABLE_ENTRY returned by the previous call to
--                     FROM_CLAUSE.NEXT_TABLE if this is not the first call
--                     for the current from clause, or the TABLE_ENTRY
--                     returned by the call to FROM_CLAUSE.TABLES_AT_-
--                     CURRENT_SCOPE if this is the first call for the
--                     current from clause.
--            Returns: The TABLE_ENTRY to use in the next call to FROM_-
--                     CLAUSE.NEXT_TABLE (valid only if second parameter is
--                     returned TRUE)
--   (out) TRUE if there are additional tables mentioned in this from
--         clause, FALSE otherwise
--   (out) The ACCESS_TYPE_DESCRIPTOR for the current table in the from
--         clause.  (When processing *, we don't care about whether the
--         table is exposed or is referenced with a correlation name.)
```

```
  procedure NEXT_TABLE
            ( CURRENT_ENTRY : in out TABLE_ENTRY;
              MORE_ENTRIES  : out    BOOLEAN;
              TABLE         : out    DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR );

private

  type TABLE_ENTRY_RECORD ( IS_CORRELATED : BOOLEAN );

  type TABLE_ENTRY is access TABLE_ENTRY_RECORD;

  type TABLE_ENTRY_RECORD ( IS_CORRELATED : BOOLEAN ) is
    record
      NEXT_TABLE : TABLE_ENTRY;
      case IS_CORRELATED is
        when TRUE =>
          CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY;
        when FALSE =>
          TABLE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      end case;
    end record;

  type INFORMATION_RECORD is
    record
      OUTER_SCOPE : INFORMATION;
      TABLE_LIST  : TABLE_ENTRY;
    end record;

end FROM_CLAUSE;
```

3.11.53 package fromb.ada

```
-- fromb.ada - internal data structures for from clauses

with CORRELATION, DDL_DEFINITIONS;
package body FROM_CLAUSE is

   use DDL_DEFINITIONS;

function AT_NEW_SCOPE
   ( SCOPE : INFORMATION )
   return INFORMATION is
begin
   return new INFORMATION_RECORD'(OUTER_SCOPE => SCOPE, TABLE_LIST => null);
end AT_NEW_SCOPE;

function AT_OUTER_SCOPE
   ( SCOPE : INFORMATION )
   return INFORMATION is
begin
```

```
   return SCOPE.OUTER_SCOPE;
end AT_OUTER_SCOPE;

procedure NAMES_EXPOSED_TABLE
   ( SCOPE : INFORMATION;
     TABLE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR ) is
begin
   SCOPE.TABLE_LIST := new TABLE_ENTRY_RECORD'
                              (IS_CORRELATED => FALSE,
                               NEXT_TABLE    => SCOPE.TABLE_LIST,
                               TABLE         => TABLE);
end NAMES_EXPOSED_TABLE;

procedure NAMES_CORRELATED_TABLE
   ( SCOPE            : INFORMATION;
     CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY ) is
begin
   SCOPE.TABLE_LIST := new TABLE_ENTRY_RECORD'
                              (IS_CORRELATED    => TRUE,
                               NEXT_TABLE       => SCOPE.TABLE_LIST,
                               CORRELATION_NAME => CORRELATION_NAME);
end NAMES_CORRELATED_TABLE;

function FIND_NAME_IN_TABLE_LIST
   ( NAME       : STRING;
     TABLE_LIST : TABLE_ENTRY )
   return TABLE_ENTRY is
   CURRENT_TABLE : TABLE_ENTRY := TABLE_LIST;
begin
   while CURRENT_TABLE /= null and then
      (        -- not matched correlated name
       (CURRENT_TABLE.IS_CORRELATED and then
        NAME /= STRING(CORRELATION.NAME_DECLARED_FOR
                       (CURRENT_TABLE.CORRELATION_NAME).all))
      or else  -- not matched table name
       (not CURRENT_TABLE.IS_CORRELATED and then
        NAME /= STRING(CURRENT_TABLE.TABLE.FULL_NAME.NAME.all))) loop
      CURRENT_TABLE := CURRENT_TABLE.NEXT_TABLE;
   end loop;
   return CURRENT_TABLE;
end FIND_NAME_IN_TABLE_LIST;

procedure EXPOSES_NAME
   ( NAME             : in  STRING;
     SCOPE            : in  INFORMATION;
     THIS_SCOPE_ONLY  : in  BOOLEAN;
     EXPOSED_TABLE    : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
     CORRELATION_NAME : out CORRELATION.NAME_DECLARED_ENTRY ) is
   CURRENT_SCOPE : INFORMATION := SCOPE;
```

```
   TABLE          : TABLE_ENTRY;
begin
   EXPOSED_TABLE    := null;
   CORRELATION_NAME := null;
   while CURRENT_SCOPE /= null loop
      TABLE := FIND_NAME_IN_TABLE_LIST (NAME, CURRENT_SCOPE.TABLE_LIST);
      exit when TABLE /= null or else THIS_SCOPE_ONLY;
      CURRENT_SCOPE := CURRENT_SCOPE.OUTER_SCOPE;
   end loop;
   if TABLE /= null then
      if TABLE.IS_CORRELATED then
         CORRELATION_NAME := TABLE.CORRELATION_NAME;
      else
         EXPOSED_TABLE := TABLE.TABLE;
      end if;
   end if;
end EXPOSES_NAME;

function FIND_COLUMN_IN_TABLE
   ( NAME  : STRING;
     TABLE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR )
   return DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR is
   CURRENT_COMPONENT : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                             TABLE.FIRST_COMPONENT;
begin
   while CURRENT_COMPONENT /= null and then
      NAME /= STRING(CURRENT_COMPONENT.FULL_NAME.NAME.all) loop
      CURRENT_COMPONENT := CURRENT_COMPONENT.NEXT_ONE;
   end loop;
   if CURRENT_COMPONENT /= null then
      return CURRENT_COMPONENT.FULL_NAME;
   else
      return null;
   end if;
end FIND_COLUMN_IN_TABLE;

function GET_TABLE_IN_TABLE_ENTRY
   ( TABLE_ENT : TABLE_ENTRY )
   return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
begin
   if TABLE_ENT.IS_CORRELATED then
      return CORRELATION.TABLE_DECLARED_FOR (TABLE_ENT.CORRELATION_NAME);
   else
      return TABLE_ENT.TABLE;
   end if;
end GET_TABLE_IN_TABLE_ENTRY;

procedure MAKES_COLUMN_VISIBLE
   ( NAME            : in STRING;
```

```
      SCOPE           : in INFORMATION;
      THIS_SCOPE_ONLY : in BOOLEAN;
      COLUMN_APPEARS_IN_MORE_THAN_ONE_TABLE : out BOOLEAN;
      DESCRIPTOR      : out DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR ) is
   CURRENT_SCOPE : INFORMATION := SCOPE;
   FOUND_COLUMN  : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
begin
   COLUMN_APPEARS_IN_MORE_THAN_ONE_TABLE := FALSE;
   DESCRIPTOR := null;
   while CURRENT_SCOPE /= null loop
      declare
         CURRENT_TABLE_ENTRY : TABLE_ENTRY := CURRENT_SCOPE.TABLE_LIST;
      begin
         while CURRENT_TABLE_ENTRY /= null loop
            -- must search all tables in list to check for duplicates.
            declare
               COLUMN_IN_TABLE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                    FIND_COLUMN_IN_TABLE
                                       (NAME,
                                        GET_TABLE_IN_TABLE_ENTRY
                                           (CURRENT_TABLE_ENTRY));
            begin
               if COLUMN_IN_TABLE /= null then
                  if FOUND_COLUMN /= null then
                     COLUMN_APPEARS_IN_MORE_THAN_ONE_TABLE := TRUE;
                  else
                     FOUND_COLUMN := COLUMN_IN_TABLE;
                  end if;
               end if;
            end;
            CURRENT_TABLE_ENTRY := CURRENT_TABLE_ENTRY.NEXT_TABLE;
         end loop;
   exit when FOUND_COLUMN /= null or else THIS_SCOPE_ONLY;
      end;
      CURRENT_SCOPE := CURRENT_SCOPE.OUTER_SCOPE;
   end loop;
   DESCRIPTOR := FOUND_COLUMN;
end MAKES_COLUMN_VISIBLE;

function TABLES_AT_CURRENT_SCOPE
   (SCOPE : INFORMATION)
   return TABLE_ENTRY is
begin
   return SCOPE.TABLE_LIST;
end TABLES_AT_CURRENT_SCOPE;

procedure NEXT_TABLE
   (CURRENT_ENTRY : in out TABLE_ENTRY;
    MORE_ENTRIES  : out    BOOLEAN;
```

```
    TABLE          : out    DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR) is
begin
   if CURRENT_ENTRY /= null then
      MORE_ENTRIES  := CURRENT_ENTRY.NEXT_TABLE /= null;
      TABLE         := GET_TABLE_IN_TABLE_ENTRY (CURRENT_ENTRY);
      CURRENT_ENTRY := CURRENT_ENTRY.NEXT_TABLE;
   else
      -- this is really a system error since we assume that CURRENT_ENTRY
      -- designates a valid table.
      MORE_FNTRIES  := FALSE;
      TABLE         := null;
   end if;
end NEXT_TABLE;

end FROM_CLAUSE;
```

3.11.54 package clauses.ada

```
with FROM_CLAUSE;

package CLAUSE is

  procedure PROCESS_FROM_CLAUSE
           (SCOPE : FROM_CLAUSE.INFORMATION);

end CLAUSE;
```

3.11.55 package clauseb.ada

```
with LEXICAL_ANALYZER, FROM_CLAUSE, DDL_DEFINITIONS, TABLE, CORRELATION,
     UNQUALIFIED_NAME;
use  LEXICAL_ANALYZER, DDL_DEFINITIONS, CORRELATION;

package body CLAUSE is

--------------------------------------------------------------------------
-- GOT_FROM_AMPERSAND - read token and gobble it and return true if it's &
--                      otherwise return false

  function GOT_FROM_AMPERSAND
           return BOOLEAN is

    AMPERSAND_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    AMPERSAND_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if AMPERSAND_TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       AMPERSAND_TOKEN.DELIMITER = LEXICAL_ANALYZER.AMPERSAND then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      return TRUE;
```

```
    else
      return FALSE;
    end if;
  end GOT_FROM_AMPERSAND;

--------------------------------------------------------------------------------
-- PROCESS_TABLE_REFERENCE -

  procedure PROCESS_TABLE_REFERENCE
          (SCOPE               : FROM_CLAUSE.INFORMATION;
           RETURNS_TABLE_LIST  : BOOLEAN;
           TABLE_TOKEN         : LEXICAL_ANALYZER.LEXICAL_TOKEN;
           CORRELATION_TOKEN   : LEXICAL_ANALYZER.LEXICAL_TOKEN) is

    TABLE_DES              : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    STATUS                 : CORRELATION.NAME_REFERENCE_STATUS;
    CORRELATION_NAME       : CORRELATION.NAME_DECLARED_ENTRY;
    DUMMY_TABLE            : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    DUMMY_CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY;
    TABLE_STATUS           : TABLE.NAME_STATUS;

  begin
    TABLE.DESCRIPTOR_FOR (TABLE_TOKEN.ID.all, TABLE_STATUS, TABLE_DES);
    case TABLE_STATUS is
      when TABLE.NAME_UNDEFINED =>
                LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
                "Table name is undefined");
      when TABLE.NAME_AMBIGUOUS =>
                LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
                "Table name is ambiguous");
      when TABLE.NAME_UNIQUE    => null;
    end case;
    if CORRELATION_TOKEN = null then
      FROM_CLAUSE.EXPOSES_NAME (TABLE_TOKEN.ID.all, SCOPE, TRUE, DUMMY_TABLE,
                                DUMMY_CORRELATION_NAME);
      if DUMMY_TABLE /= null or else DUMMY_CORRELATION_NAME /= null then
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
        "Table name already used in from clause");
      end if;
      if RETURNS_TABLE_LIST then
        UNQUALIFIED_NAME.RETURNS_TABLE_LIST (TABLE_DES.FULL_NAME.NAME);
      else
        UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DES.FULL_NAME.NAME);
      end if;
    else
      if RETURNS_TABLE_LIST then
        CORRELATION.NAME_RETURNS_TABLE_LIST (CORRELATION_TOKEN.ID.all,
                    TABLE_DES, STATUS, CORRELATION_NAME);
      else
```

```
        CORRELATION.NAME_RETURNS_TABLE_NAME (CORRELATION_TOKEN.ID.all,
                    TABLE_DES, STATUS, CORRELATION_NAME);
      end if;
      case STATUS is
        when CORRELATION.NAME_VALID => null;
        when CORRELATION.NAME_NOT_DECLARED =>
            LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
            "Correlation name has not been declared");
        when CORRELATION.NAME_DECLARED_FOR_DIFFERENT_TABLE =>
            LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
            "Correlation name has already been declared for another table");
      end case;
      FROM_CLAUSE.EXPOSES_NAME (CORRELATION_TOKEN.ID.all, SCOPE, TRUE,
                  DUMMY_TABLE, DUMMY_CORRELATION_NAME);
      if DUMMY_TABLE /= null or else DUMMY_CORRELATION_NAME /= null then
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
        "Correlation name has already been declared for another table");
      end if;
      FROM_CLAUSE.NAMES_CORRELATED_TABLE (SCOPE, CORRELATION_NAME);
    end if;
  end PROCESS_TABLE_REFERENCE;

--------------------------------------------------------------------
-- GOT_FROM_TABLE - reads tokens for a table or correlation.table and
--                  processes them accordingly.  Return true after one
--                  is successfully processed.

  function GOT_FROM_TABLE
          (SCOPE       : FROM_CLAUSE.INFORMATION;
           FIRST_TABLE : BOOLEAN)
           return        BOOLEAN is

    TABLE_TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    DOT_TOKEN         : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    CORRELATION_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    CORRELATION_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if CORRELATION_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
      "Expecting table name");
    else
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      DOT_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if DOT_TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         DOT_TOKEN.DELIMITER = LEXICAL_ANALYZER.DOT then
        TABLE_TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
        if TABLE_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER then
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
```

```
          "Expecting correlation_name.table_name");
        end if;
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        TABLE_TOKEN := CORRELATION_TOKEN;
        CORRELATION_TOKEN := null;
        DOT_TOKEN := null;
      end if;
    end if;
    PROCESS_TABLE_REFERENCE (SCOPE, FIRST_TABLE, TABLE_TOKEN,
                             CORRELATION_TOKEN);
    return TRUE;
  end GOT_FROM_TABLE;

----------------------------------------------------------------------------
-- GOT_FROM_CLAUSE - we should now find FROM => tokens.  If not print
--                   error message.  If we do return true

  function GOT_FROM_CLAUSE
           return BOOLEAN is

    FROM_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;

  begin
    FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    case FROM_TOKEN.KIND is
      when LEXICAL_ANALYZER.IDENTIFIER        =>
        if FROM_TOKEN.ID.all = "FROM" then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
          FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
          case FROM_TOKEN.KIND is
            when LEXICAL_ANALYZER.DELIMITER  =>
                  if FROM_TOKEN.DELIMITER = LEXICAL_ANALYZER.ARROW then
                    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
                    return TRUE;
                  end if;
            when others   =>  null;
          end case;
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (FROM_TOKEN,
          "Expecting token: =>");
        end if;
      when others            => null;
    end case;
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (FROM_TOKEN,
    "Expecting token: FROM");
  end GOT_FROM_CLAUSE;

----------------------------------------------------------------------------
```

```
-- PROCESS_FROM_CLAUSE - process a from clause

  procedure PROCESS_FROM_CLAUSE
          (SCOPE : FROM_CLAUSE.INFORMATION) is

    FIRST_TABLE : BOOLEAN := TRUE;

  begin
    if GOT_FROM_CLAUSE then
      loop
        exit when not GOT_FROM_TABLE (SCOPE, FIRST_TABLE);
        exit when not GOT_FROM_AMPERSAND;
      end loop;
    end if;
  end PROCESS_FROM_CLAUSE;
end CLAUSE;
```

3.11.56 package indics.ada

```
-- indics.ada - post process data structures for INDICATOR functions

with DDL_DEFINITIONS;
package INDICATOR is

-- Although this implementation does not support NULL database values,
-- INDICATOR functions are still required in occasional contexts to force a
-- particular interpretation on part of an Ada/SQL statement.  Example (A and
-- B are program values of type BOOLEAN):
--
--   SELEC ( A & B , ...                returns one column (constant with
--                                      respect to the database)
--
--   SELEC ( INDICATOR ( A ) & B , ... returns two columns (constant with
--                                      respect to the database)

-- We only recognize INDICATOR functions with a single parameter.

-- Depending on the context, INDICATOR may return either a strongly typed
-- database value (based on the program type of its parameter), or an untyped
-- database value (SQL_OBJECT).

-- The generated INDICATOR function returning a strongly typed result is:
--
--   function INDICATOR is new
--    ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
--    ( fully_qualified_type_name,
--      ADA_SQL.package_TYPE_PACKAGE.type_simple_name_TYPE );

-- The generated INDICATOR function returning an untyped result is:
--
```

```
--   function INDICATOR is new
--    ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION
--    ( fully_qualified_type_name , ADA_SQL_FUNCTIONS.SQL_OBJECT );

-- The following notations are used in the above:
--
-- package = name of the library unit in which the program type is declared
--
-- type_simple_name = simple name of the program type
--
-- fully_qualified_type_name is of the form package.ADA_SQL.type_simple_name
-- if the type is declared in a DDL package, or of the form package.type_-
-- simple_name if the type is declared in a predefined package

-- All the information required to generate either kind of INDICATOR function
-- for a particular type is contained within that type's ACCESS_FULL_NAME_-
-- DESCRIPTOR, which is used by the routines defined here to identify a given
-- type.

-- INDICATOR.RETURNS_STRONGLY_TYPED and INDICATOR.RETURNS_SQL_OBJECT are
-- called to remember that the particular kind of INDICATOR function must be
-- generated for the given type.  They automatically ignore duplicate
-- requests.

  procedure RETURNS_STRONGLY_TYPED
            ( PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

  procedure RETURNS_SQL_OBJECT
            ( PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR );

-- INDICATOR.POST_PROCESSING causes the generated INDICATOR functions to be
-- produced.

  procedure POST_PROCESSING;

end INDICATOR;
```

3.11.57 package indicb.ada

```
-- indicb.ada - post process data structures for INDICATOR functions

with TEXT_PRINT, DDL_DEFINITIONS, DUMMY, PROGRAM_CONVERSION, DATABASE_TYPE;
use  TEXT_PRINT;
package body INDICATOR is

   use DDL_DEFINITIONS;

   type INDICATOR_ENTRY_RECORD;
   type INDICATOR_ENTRY is access INDICATOR_ENTRY_RECORD;
```

```
   type INDICATOR_ENTRY_RECORD is
      record
         PROGRAM_TYPE             : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                       DUMMY.ACCESS_FULL_NAME_DESCRIPTOR;
         RETURNS_SQL_OBJECT       : BOOLEAN := FALSE;
         RETURNS_STRONGLY_TYPED : BOOLEAN := FALSE;
         NEXT_INDICATOR           : INDICATOR_ENTRY;
      end record;

   INDICATOR_LIST : INDICATOR_ENTRY := new INDICATOR_ENTRY_RECORD;

function ">="
   (LEFT, RIGHT : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return BOOLEAN is
begin
   if LEFT.FULL_PACKAGE_NAME.all > RIGHT.FULL_PACKAGE_NAME.all then
      return TRUE;
   elsif LEFT.FULL_PACKAGE_NAME /= RIGHT.FULL_PACKAGE_NAME then
      return FALSE;
   elsif LEFT.NAME.all >= RIGHT.NAME.all then
      return TRUE;
   else
      return FALSE;
   end if;
end ">=";

function NEW_INDICATOR
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
   return INDICATOR_ENTRY is
   TRACER : INDICATOR_ENTRY := INDICATOR_LIST;
   RESULT : INDICATOR_ENTRY;
begin
   while TRACER.NEXT_INDICATOR /= null and then
      PROGRAM_TYPE >= TRACER.NEXT_INDICATOR.PROGRAM_TYPE loop
      TRACER := TRACER.NEXT_INDICATOR;
   end loop;
   if PROGRAM_TYPE = TRACER.PROGRAM_TYPE then
      RESULT := TRACER;
   else
      RESULT := new INDICATOR_ENTRY_RECORD;
      RESULT.PROGRAM_TYPE := PROGRAM_TYPE;
      RESULT.NEXT_INDICATOR := TRACER.NEXT_INDICATOR;
      TRACER.NEXT_INDICATOR := RESULT;
   end if;
   PROGRAM_CONVERSION.REQUIRED_FOR (PROGRAM_TYPE);
   return RESULT;
end NEW_INDICATOR;

procedure RETURNS_STRONGLY_TYPED
```

```
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_INDICATOR : INDICATOR_ENTRY := NEW_INDICATOR (PROGRAM_TYPE);
begin
   OUR_INDICATOR.RETURNS_STRONGLY_TYPED := TRUE;
   DATABASE_TYPE.REQUIRED_FOR (PROGRAM_TYPE);
end RETURNS_STRONGLY_TYPED;

procedure RETURNS_SQL_OBJECT
   (PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   OUR_INDICATOR : INDICATOR_ENTRY := NEW_INDICATOR (PROGRAM_TYPE);
begin
   OUR_INDICATOR.RETURNS_SQL_OBJECT := TRUE;
end RETURNS_SQL_OBJECT;

procedure POST_PROCESSING is
   TRACER : INDICATOR_ENTRY := INDICATOR_LIST.NEXT_INDICATOR;
begin
   while TRACER /= null loop
      if TRACER.RETURNS_SQL_OBJECT then
         SET_INDENT (2);
         PRINT ("function INDICATOR is new");
         PRINT_LINE;
         SET_INDENT (4);
         PRINT ("ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION");
         PRINT_LINE;
         SET_INDENT (6);
         PRINT ("( ");
         PRINT (STRING(TRACER.PROGRAM_TYPE.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.PROGRAM_TYPE.NAME.all));
         PRINT (",");
         PRINT_LINE;
         SET_INDENT (8);
         PRINT ("ADA_SQL.SQL_OBJECT );");
         PRINT_LINE;
         BLANK_LINE;
      end if;
      if TRACER.RETURNS_STRONGLY_TYPED then
         SET_INDENT (2);
         PRINT ("function INDICATOR is new");
         PRINT_LINE;
         SET_INDENT (4);
         PRINT ("ADA_SQL_FUNCTIONS.INDICATOR_FUNCTION");
         PRINT_LINE;
         SET_INDENT (6);
         PRINT ("( ");
         PRINT (STRING(TRACER.PROGRAM_TYPE.FULL_PACKAGE_NAME.all) & ".");
         PRINT (STRING(TRACER.PROGRAM_TYPE.NAME.all));
         PRINT (",");
         PRINT_LINE;
```

```
            SET_INDENT (8);
            PRINT ("ADA_SQL.");
            PRINT (STRING(TRACER.PROGRAM_TYPE.SCHEMA_UNIT.NAME.all) &
                   "_TYPE_PACKAGE.");
            PRINT (STRING(TRACER.PROGRAM_TYPE.NAME.all) & "_TYPE ");
            PRINT (");");
            PRINT_LINE;
            BLANK_LINE;
         end if;
         TRACER := TRACER.NEXT_INDICATOR;
      end loop;
   end POST_PROCESSING;

end INDICATOR;
```

3.11.58 package genfuncs.ada

```
-- genfuncs.ada -- post process/info for expression-type unary & binary ops

with ADA_SQL_FUNCTION_DEFINITIONS, DDL_DEFINITIONS;
 use ADA_SQL_FUNCTION_DEFINITIONS, DDL_DEFINITIONS;
package GENERATED_FUNCTIONS is

-- Two basic kinds of expression-related functions are generated by the
-- application scanner:
--
-- (1) Unary
--
-- (2) Binary

-- In order to generate the functions, the types of their operands (single
-- operands for unary functions, left and right operands for binary functions)
-- and results must be known.  There are six kinds of operand/result type used
-- with expression-type operators (see type OPERAND_KIND):
--
-- (1) Insert item - used with "<=" and "and" operators for building insert
--     value lists
--
-- (2) SQL object - used with many operators for database values where type is
--     not important to semantics
--
-- (3) Table list - left operand and result type for "&" operator used in
--     building from clauses
--
-- (4) Table name - right operand for "&" operator used in building from
--     clauses
--
-- (5) Typed SQL object - used for database values where typing is important
--     to the semantics; the actual operand/result type is declared in the
--     generated package, and is related to the declaration of the
```

```
--     corresponding program type
--
-- (6) User type - program types may be operands only (e.g., COLUMN + 2); no
--     SQL operator returns a program type defined by the user

-- Operand kinds (1) - (4) require no additional information for complete
-- specification; each has a unique operand/result type.  The specific
-- operand/result type for operand kinds (5) and (6) is, however, dependent on
-- the user-defined program type.  This information is passed to the routines
-- visible here as a pointer to the ACCESS_FULL_NAME_DESCRIPTOR for the
-- program type.  When indicating an operand of kinds (1) - (4) to these
-- routines, the corresponding ACCESS_FULL_NAME_DESCRIPTOR parameter must be
-- null.

-- The routines visible here maintain data structures remembering the above
-- information for all expression-type functions required.  See the package
-- body for details on the data structures.

-- Note: Functions are "added" whenever encountered; the routines
-- automatically avoid generating duplicates.

  type OPERAND_KIND is ( O_INSERT_ITEM , O_SQL_OBJECT , O_TABLE_LIST ,
   O_TABLE_NAME , O_TYPED_SQL_OBJECT , O_USER_TYPE );

  procedure ADD_UNARY_FUNCTION
            ( OPERATION      : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
              PARAMETER_KIND : OPERAND_KIND;
              PARAMETER      : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                null;
              RESULT_KIND    : OPERAND_KIND;
              RESULT         : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                null );

  procedure ADD_BINARY_FUNCTION
            ( OPERATION :
               ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
              LEFT_PARAMETER_KIND :
               OPERAND_KIND;
              LEFT_PARAMETER :
               DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
              RIGHT_PARAMETER_KIND :
               OPERAND_KIND;
              RIGHT_PARAMETER :
               DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
              RESULT_KIND :
               OPERAND_KIND;
              RESULT :
               DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null );
```

```
-- Post processing to generate expression-type functions is quite simple:
-- UNARY_OPERATION or BINARY_OPERATION is instantiated, as appropriate, with
-- the required operand and result types.  See the package body for details on
-- code generated; the visible routine causes post processing to be performed.

  procedure POST_PROCESSING;

end GENERATED_FUNCTIONS;
```

3.11.59 package genfuncb.ada

```
-- genfuncb.ada -- post process/info for expression-type unary & binary ops

with TEXT_PRINT, DATABASE_TYPE, PROGRAM_CONVERSION;
 use TEXT_PRINT;
package body GENERATED_FUNCTIONS is

  type OPERATION_KIND is ( UNARY , BINARY );

  type OPERAND_DESCRIPTOR ( KIND : OPERAND_KIND := O_USER_TYPE ) is
    record
      case KIND is
        when O_INSERT_ITEM .. O_TABLE_NAME =>
          null;
        when O_TYPED_SQL_OBJECT .. O_USER_TYPE =>
          USER_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
      end case;
    end record;

  type FUNCTION_LIST_RECORD ( KIND : OPERATION_KIND );

  type FUNCTION_LIST is access FUNCTION_LIST_RECORD;

  type FUNCTION_LIST_RECORD ( KIND : OPERATION_KIND ) is
    record
      NEXT      : FUNCTION_LIST;
      OPERATION : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
      RESULT    : OPERAND_DESCRIPTOR;
      case KIND is
        when UNARY =>
          OPERAND : OPERAND_DESCRIPTOR;
        when BINARY =>
          LEFT_OPERAND  : OPERAND_DESCRIPTOR;
          RIGHT_OPERAND : OPERAND_DESCRIPTOR;
      end case;
    end record;

  FUNCTIONS : FUNCTION_LIST := new
   FUNCTION_LIST_RECORD'
   ( KIND => UNARY,
```

```
        NEXT => null,
        OPERATION => ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,
        RESULT => ( KIND => O_SQL_OBJECT ),
        OPERAND => ( KIND => O_SQL_OBJECT ) );

type PRINT_NAME_STRING is new STRING;

type PRINT_NAME is access PRINT_NAME_STRING;

type PRINT_NAME_ARRAY is
 array ( ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION range <> )
  of PRINT_NAME;

PRINT_NAMES : constant PRINT_NAME_ARRAY :=
 ( O_AVG               => new PRINT_NAME_STRING' ( "AVG" ),
   O_MAX               => new PRINT_NAME_STRING' ( "MAX" ),
   O_MIN               => new PRINT_NAME_STRING' ( "MIN" ),
   O_SUM               => new PRINT_NAME_STRING' ( "SUM" ),
   O_UNARY_PLUS        => new PRINT_NAME_STRING' ( """+""" ),
   O_UNARY_MINUS       => new PRINT_NAME_STRING' ( """-""" ),
   O_PLUS              => new PRINT_NAME_STRING' ( """+""" ),
   O_MINUS             => new PRINT_NAME_STRING' ( """-""" ),
   O_TIMES             => new PRINT_NAME_STRING' ( """*""" ),
   O_DIVIDE            => new PRINT_NAME_STRING' ( """/""" ),
   O_EQ                => new PRINT_NAME_STRING' ( "EQ" ),
   O_NE                => new PRINT_NAME_STRING' ( "NE" ),
   O_LT                => new PRINT_NAME_STRING' ( """<""" ),
   O_GT                => new PRINT_NAME_STRING' ( """>""" ),
   O_LE                => new PRINT_NAME_STRING' ( """<=""" ),
   O_GE                => new PRINT_NAME_STRING' ( """>=""" ),
   O_BETWEEN           => new PRINT_NAME_STRING' ( "BETWEEN" ),
   O_AND               => new PRINT_NAME_STRING' ( """and""" ),
   O_IS_IN             => new PRINT_NAME_STRING' ( "IS_IN" ),
   O_OR                => new PRINT_NAME_STRING' ( """or""" ),
   O_NOT               => new PRINT_NAME_STRING' ( """not""" ),
   O_LIKE              => new PRINT_NAME_STRING' ( "LIKE" ),
   O_AMPERSAND         => new PRINT_NAME_STRING' ( """&""" ),
   O_SELEC             => null,
   O_SELECT_DISTINCT   => null,
   O_ASC               => new PRINT_NAME_STRING' ( "ASC" ),
   O_DESC              => new PRINT_NAME_STRING' ( "DESC" ),
   O_TABLE_COLUMN_LIST => null,
   O_COUNT_STAR        => null,
   O_NULL_OP           => new PRINT_NAME_STRING' ( "" ),
   O_STAR              => null,
   O_NOT_IN            => new PRINT_NAME_STRING' ( "NOT_IN" ),
   O_VALUES            => null,
   O_DECLAR            => null );
```

```
type COMPARISON_RESULT is ( LESS_THAN , EQUAL , GREATER_THAN );

function COMPARE ( LEFT, RIGHT : OPERAND_DESCRIPTOR )
  return COMPARISON_RESULT is
begin
  if LEFT.KIND = RIGHT.KIND then
    if LEFT.KIND in O_TYPED_SQL_OBJECT .. O_USER_TYPE then
      if LEFT.USER_TYPE.SCHEMA_UNIT.NAME.all >
         RIGHT.USER_TYPE.SCHEMA_UNIT.NAME.all then
        return GREATER_THAN;
      elsif LEFT.USER_TYPE.SCHEMA_UNIT.NAME.all <
            RIGHT.USER_TYPE.SCHEMA_UNIT.NAME.all then
        return LESS_THAN;
      elsif LEFT.USER_TYPE.NAME.all > RIGHT.USER_TYPE.NAME.all then
        return GREATER_THAN;
      elsif LEFT.USER_TYPE.NAME.all < RIGHT.USER_TYPE.NAME.all then
        return LESS_THAN;
      else
        return EQUAL;
      end if;
    else
      return EQUAL;
    end if;
  elsif LEFT.KIND < RIGHT.KIND then
    return LESS_THAN;
  else
    return GREATER_THAN;
  end if;
end COMPARE;

function COMPARE ( LEFT, RIGHT : FUNCTION_LIST ) return COMPARISON_RESULT is
begin
  if LEFT.OPERATION /= RIGHT.OPERATION then
    if PRINT_NAMES(LEFT.OPERATION).all > PRINT_NAMES(RIGHT.OPERATION).all
     then
      return GREATER_THAN;
    elsif PRINT_NAMES(LEFT.OPERATION).all < PRINT_NAMES(RIGHT.OPERATION).all
     then
      return LESS_THAN;
    else
      if LEFT.KIND = RIGHT.KIND then
        raise CONSTRAINT_ERROR;
      end if;
      if LEFT.KIND = UNARY then
        return LESS_THAN;
      else
        return GREATER_THAN;
      end if;
    end if;
```

```
      end if;
      if LEFT.KIND /= RIGHT.KIND then
        if LEFT.KIND = UNARY then
          return LESS_THAN;
        else
          return GREATER_THAN;
        end if;
      end if;
      case COMPARE (LEFT.RESULT, RIGHT.RESULT) is
        when LESS_THAN =>
          return LESS_THAN;
        when GREATER_THAN =>
          return GREATER_THAN;
        when EQUAL =>
          if LEFT.KIND = UNARY then
            case COMPARE (LEFT.OPERAND, RIGHT.OPERAND) is
              when LESS_THAN =>
                return LESS_THAN;
              when GREATER_THAN =>
                return GREATER_THAN;
              when EQUAL =>
                return EQUAL;
            end case;
          else
            case COMPARE (LEFT.LEFT_OPERAND, RIGHT.LEFT_OPERAND) is
              when LESS_THAN =>
                return LESS_THAN;
              when GREATER_THAN =>
                return GREATER_THAN;
              when EQUAL =>
                case COMPARE (LEFT.RIGHT_OPERAND, RIGHT.RIGHT_OPERAND) is
                  when LESS_THAN =>
                    return LESS_THAN;
                  when GREATER_THAN =>
                    return GREATER_THAN;
                      when EQUAL =>
                        return EQUAL;
                end case;
            end case;
          end if;
      end case;
    end COMPARE;

    procedure ADD_FUNCTION ( NEW_FUNCTION : FUNCTION_LIST ) is
      CURRENT_FUNCTION : FUNCTION_LIST := FUNCTIONS;
      COMPARISON       : COMPARISON_RESULT;
    begin
      while CURRENT_FUNCTION.NEXT /= null loop
        COMPARISON := COMPARE ( NEW_FUNCTION , CURRENT_FUNCTION.NEXT );
```

```
    exit when COMPARISON = LESS_THAN;
      if COMPARISON = EQUAL then
        return;
      end if;
      CURRENT_FUNCTION := CURRENT_FUNCTION.NEXT;
    end loop;
    NEW_FUNCTION.NEXT := CURRENT_FUNCTION.NEXT;
    CURRENT_FUNCTION.NEXT := NEW_FUNCTION;
  end ADD_FUNCTION;

  function BUILD_OPERAND_DESCRIPTOR
           ( KIND : OPERAND_KIND ;
             NAME : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR )
   return OPERAND_DESCRIPTOR is
  begin
    if KIND in O_INSERT_ITEM .. O_TABLE_NAME then
      if NAME /= null then
        raise CONSTRAINT_ERROR;
      end if;
    else
      if NAME = null then
        raise CONSTRAINT_ERROR;
      end if;
    end if;
    case KIND is
      when O_INSERT_ITEM      => return ( KIND => O_INSERT_ITEM );
      when O_SQL_OBJECT       => return ( KIND => O_SQL_OBJECT );
      when O_TABLE_LIST       => return ( KIND => O_TABLE_LIST );
      when O_TABLE_NAME       => return ( KIND => O_TABLE_NAME );
      when O_TYPED_SQL_OBJECT => DATABASE_TYPE.REQUIRED_FOR ( NAME );
                                 return ( O_TYPED_SQL_OBJECT , NAME );
      when O_USER_TYPE        => PROGRAM_CONVERSION.REQUIRED_FOR ( NAME );
                                 return ( O_USER_TYPE , NAME );
    end case;
  end BUILD_OPERAND_DESCRIPTOR;

  procedure ADD_UNARY_FUNCTION
            ( OPERATION      : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
              PARAMETER_KIND : OPERAND_KIND;
              PARAMETER      : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                null;
              RESULT_KIND    : OPERAND_KIND;
              RESULT         : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                                null ) is
  begin
    ADD_FUNCTION (
     new FUNCTION_LIST_RECORD'
     ( KIND =>
        UNARY,
```

```
      NEXT =>
       null,
      OPERATION =>
       OPERATION,
      RESULT =>
       BUILD_OPERAND_DESCRIPTOR ( RESULT_KIND , RESULT ),
      OPERAND =>
       BUILD_OPERAND_DESCRIPTOR ( PARAMETER_KIND , PARAMETER ) ) );
end ADD_UNARY_FUNCTION;

procedure ADD_BINARY_FUNCTION
          ( OPERATION :
             ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
            LEFT_PARAMETER_KIND :
             OPERAND_KIND;
            LEFT_PARAMETER :
             DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
            RIGHT_PARAMETER_KIND :
             OPERAND_KIND;
            RIGHT_PARAMETER :
             DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
            RESULT_KIND :
             OPERAND_KIND;
            RESULT :
             DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null ) is
begin
  ADD_FUNCTION (
   new FUNCTION_LIST_RECORD'
   ( KIND =>
       BINARY,
      NEXT =>
       null,
      OPERATION =>
       OPERATION,
      RESULT =>
       BUILD_OPERAND_DESCRIPTOR ( RESULT_KIND , RESULT ),
      LEFT_OPERAND =>
       BUILD_OPERAND_DESCRIPTOR ( LEFT_PARAMETER_KIND , LEFT_PARAMETER ),
      RIGHT_OPERAND =>
       BUILD_OPERAND_DESCRIPTOR
       ( RIGHT_PARAMETER_KIND , RIGHT_PARAMETER ) ) );
end ADD_BINARY_FUNCTION;

procedure PRINT_OPERAND ( OPERAND : OPERAND_DESCRIPTOR ) is
begin
  case OPERAND.KIND is
    when O_INSERT_ITEM =>
      PRINT ( "ADA_SQL_FUNCTIONS.INSERT_ITEM" );
    when O_SQL_OBJECT =>
```

```
        PRINT ( "ADA_SQL_FUNCTIONS.SQL_OBJECT" );
      when O_TABLE_LIST =>
        PRINT ( "ADA_SQL_FUNCTIONS.TABLE_LIST" );
      when O_TABLE_NAME =>
        PRINT ( "ADA_SQL_FUNCTIONS.TABLE_NAME" );
      when O_TYPED_SQL_OBJECT =>
        PRINT ( "ADA_SQL." );
        PRINT
        ( STRING ( OPERAND.USER_TYPE.SCHEMA_UNIT.NAME.all ) &
           "_TYPE_PACKAGE." );
        PRINT ( STRING ( OPERAND.USER_TYPE.NAME.all ) & "_TYPE" );
      when O_USER_TYPE =>
        PRINT ( STRING ( OPERAND.USER_TYPE.FULL_PACKAGE_NAME.all ) & "." );
        PRINT ( STRING ( OPERAND.USER_TYPE.NAME.all ) );
    end case;
  end PRINT_OPERAND;

  procedure POST_PROCESSING is
    CURRENT_FUNCTION : FUNCTION_LIST := FUNCTIONS.NEXT;
  begin
    while CURRENT_FUNCTION /= null loop
      SET_INDENT (2);
      PRINT ( "function " );
      PRINT ( STRING ( PRINT_NAMES(CURRENT_FUNCTION.OPERATION).all ) );
      PRINT ( " is new" );
      PRINT_LINE;
      SET_INDENT (4);
      if CURRENT_FUNCTION.KIND = UNARY then
        PRINT ( "ADA_SQL_FUNCTIONS.UNARY_OPERATION" );
      else
        PRINT ( "ADA_SQL_FUNCTIONS.BINARY_OPERATION" );
      end if;
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ( "( ADA_SQL_FUNCTIONS." );
      PRINT
      ( ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION'IMAGE
        ( CURRENT_FUNCTION.OPERATION ) );
      PRINT ( "," );
      PRINT_LINE;
      SET_INDENT (8);
      if CURRENT_FUNCTION.KIND = UNARY then
        PRINT_OPERAND ( CURRENT_FUNCTION.OPERAND );
      else
        PRINT_OPERAND ( CURRENT_FUNCTION.LEFT_OPERAND );
        PRINT ( "," );
        PRINT_LINE;
        PRINT_OPERAND ( CURRENT_FUNCTION.RIGHT_OPERAND );
      end if;
```

```
      PRINT ( "," );
      PRINT_LINE;
      PRINT_OPERAND ( CURRENT_FUNCTION.RESULT );
      PRINT ( " );" );
      PRINT_LINE;
      BLANK_LINE;
      CURRENT_FUNCTION := CURRENT_FUNCTION.NEXT;
    end loop;
  end POST_PROCESSING;

end GENERATED_FUNCTIONS;
```

3.11.60 package selecs.ada

```
-- selecs.ada - post process data structures for various flavors of SELEC

with DDL_DEFINITIONS;
package SELEC is

-- There are three SQL flavors of SELECT: SELECT, SELECT ALL, and SELECT
-- DISTINCT.  (SELECT ALL is semantically equivalent to SELECT, but we still
-- allow for its use.) The corresponding Ada/SQL keywords are SELEC,
-- SELECT_ALL, and SELECT_DISTINCT.  When we maintain information about SELEC
-- (hereafter used to refer to any of the three flavors) subprograms to
-- generate, we keep track of the name of the subprogram via an enumeration
-- value:

  type ROUTINE_NAME is ( SELEC , SELECT_ALL, SELECT_DISTINCT );

-- The first parameter to a SELEC subprogram is the list of items being
-- selected (only one item permitted if call is as a subquery).  There are
-- four possible kinds of parameter, discussed in terms of enumeration values
-- descriptive of them and the contexts in which they would appear:

-- STAR
  -- For SELEC ( '*' ..., the parameter '*' is of type STAR_TYPE, discussed in
  -- predefs.ada.

-- SQL_OBJECT
  -- The first parameter to SELEC is untyped if (1) SELEC is a function and
  -- its context of use does not require that its return result be strongly
  -- typed according to a program type, or (2) SELEC is a procedure.

  -- Contexts for (1):
  --   Subquery within exists predicate (not implemented in this version)
  --   Any query specification

  -- Context for (2):
  --   Select statement
```

```
-- DATABASE_VALUE (strongly typed)
  -- The first parameter to SELEC is typed if SELEC is a function and its
  -- context of use requires that its return result be strongly typed
  -- according to a program type.  This is the case for subqueries used in:
  --   Comparison predicate
  --   In predicate
  --   Quantified predicate (not implemented in this version)

-- PROGRAM_VALUE (strongly typed)
  -- Any SELEC, whether returning a strongly typed or an untyped result, can
  -- be called with a program value.  Of course, since this is a constant with
  -- respect to the database, its usefulness is somewhat limited, but we
  -- support it nevertheless.

  type PARAMETER_TYPE is
       ( STAR , SQL_OBJECT , DATABASE_VALUE , PROGRAM_VALUE );

-- The generated SELEC subprograms have four possible kinds of result types,
-- discussed in terms of enumeration values descriptive of them and the
-- contexts in which they would be used.

-- SQL_OBJECT
  -- Same context as for SQL_OBJECT parameter, except that query specification
  -- within insert statement returns INSERT_ITEM, not SQL_OBJECT.

-- INSERT_ITEM
  -- Query specification within insert statement.

-- DATABASE_VALUE (strongly typed)
  -- Same context as for DATABASE_VALUE parameter type.

-- PROCEDURE_CALL
  -- SELEC for a select statement is a procedure, and so has no return type.
  -- All other SELECs are functions, with return type falling into one of the
  -- above three categories.

  type RESULT_TYPE is
       ( SQL_OBJECT , INSERT_ITEM , DATABASE_VALUE , PROCEDURE_CALL );

-- SELEC.REQUIRED_FOR is called to indicate that a SELEC subprogram must be
-- generated according to the given routine name, parameter kind, and result
-- kind.  Where the parameter and/or result is strongly typed, the ACCESS_-
-- FULL_NAME_DESCRIPTOR for the relevant program type is provided.  (Database
-- types are constructed based on their corresponding program types.  Because
-- of Ada/SQL comparability rules, the same program type will apply to the
-- parameter and result, if both are strongly typed.)  If neither parameter
-- nor result is strongly typed, then null (parameter default) is supplied for
-- the program type.  Duplicate calls are fine; they are processed so that
-- only a single version of each subprogram is generated.
```

```
procedure REQUIRED_FOR
          ( ROUTINE      : ROUTINE_NAME;
            PARAMETER    : PARAMETER_TYPE;
            RESULT       : RESULT_TYPE;
            PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                            null );

-- The following table shows the various formats of SELEC subprograms
-- generated for each combination of parameter and result type.  Numbers
-- correspond to the formats shown below; "n/a" means that the indicated
-- combination is not possible with Ada/SQL.

--                  ----------------------- Result type ----------------------
-- Parameter type   SQL_OBJECT   INSERT_ITEM   DATABASE_VALUE   PROCEDURE_CALL
-- STAR                  1            1               1                2
-- SQL_OBJECT            3            3              n/a               4
-- DATABASE_VALUE       n/a          n/a              3               n/a
-- PROGRAM_VALUE         3            3               3                4

-- The following notation is used with the formats:

  -- routine_name = the name of the routine to generate (SELEC, SELECT_ALL, or
  --  SELECT_DISTINCT), according to the ROUTINE_NAME

  -- result_type = according to the RESULT_TYPE (other than PROCEDURE_CALL):
  --   SQL_OBJECT     => ADA_SQL_FUNCTIONS.SQL_OBJECT
  --   INSERT_ITEM    => ADA_SQL_FUNCTIONS.INSERT_ITEM
  --   DATABASE_VALUE => ADA_SQL.package_TYPE_PACKAGE.type_name_TYPE, where
  --    package is the name of the library unit in which the relevant program
  --    type is defined, and type_name is the simple name of the relevant
  --    program type

  -- operation = according to the ROUTINE_NAME:
  --   SELEC           => O_SELEC
  --   SELECT_ALL      => O_SELEC
  --   SELECT_DISTINCT => O_SELECT_DISTINCT

  -- parameter_type = according to PARAMETER_TYPE (other than STAR):
  --   SQL_OBJECT     => ADA_SQL_FUNCTIONS.SQL_OBJECT
  --   DATABASE_VALUE => as with result_type DATABASE_VALUE, above
  --   PROGRAM_VALUE  => package.type_name (if package is a predefined
  --    package) or package.ADA_SQL.type_name (if package is a DDL package),
  --    where package and type_name are as with result_type DATABASE_VALUE,
  --    above

-- Format 1: SELECT * function

  -- Specification:
  --
```

```
  -- function routine_name
  --          ( WHAT     : STAR_TYPE;
  --            FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
  --            WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
  --  return result_type;

  -- Body parts:
  --
  -- function SELEC_STAR_SUBQUERY is new
  --  ADA_SQL_FUNCTIONS.STAR_SUBQUERY
  --  ( ADA_SQL_FUNCTIONS.operation , result_type );
  --
  -- function routine_name
  --          ( WHAT     : STAR_TYPE;
  --            FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
  --            WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )
  --  return result_type is
  -- begin
  --   return SELEC_STAR_SUBQUERY ( FROM , WHERE , GROUP_BY , HAVING );
  -- end routine_name;

-- Format 2: SELECT * procedure

  -- Specification:
  --
  -- procedure routine_name
  --          ( WHAT     : STAR_TYPE;
  --            FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
  --            WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --            HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                        ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT );

  -- Body parts:
  --
  -- procedure SELEC_STAR is new
  --  ADA_SQL_FUNCTIONS.STAR_SELECT ( ADA_SQL_FUNCTIONS.operation );
```

```
  --
  -- procedure routine_name
  --            ( WHAT     : STAR_TYPE;
  --              FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;
  --              WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                          ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --              GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                          ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;
  --              HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=
  --                          ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT ) is
  -- begin
  --   SELEC_STAR ( FROM , WHERE , GROUP_BY , HAVING );
  -- end routine_name;

-- Format 3: SELECT functions other than SELECT *

  -- function routine_name is new
  --  ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY
  --  ( ADA_SQL_FUNCTIONS.operation , parameter_type , result_type );

-- Format 4: SELECT procedures other than SELECT *

  -- procedure routine_name is new
  --  ADA_SQL_FUNCTIONS.SELECT_LIST_SELECT
  --  ( ADA_SQL_FUNCTIONS.operation , parameter_type );

-- Post processing to generate SELEC subprograms is done in two parts: (1) the
-- specification parts of formats 1 and 2, and all of formats 3 and 4 are
-- produced, then (2) the body parts of formats 1 and 2 are produced.  (I
-- don't know why I made formats 1 and 2 so complicated; it seems that generic
-- subprograms should have been able to handle the entire thing instead of
-- having to actually generate bodies.  Perhaps we can change this later.)
-- SELEC.POST_PROCESSING_1 and SELEC.POST_PROCESSING_2 perform these two post
-- processing steps.

  procedure POST_PROCESSING_1;

  procedure POST_PROCESSING_2;

end SELEC;
```

3.11.61 package selecb.ada

```
-- selecs.ada - post process data structures for various flavors of SELEC

with TEXT_PRINT, DDL_DEFINITIONS, PREDEFINED, DATABASE_TYPE, PROGRAM_CONVERSION;
use  TEXT_PRINT;
package body SELEC is

   use DDL_DEFINITIONS;
```

```
   type REQUIRED_SELECT_ENTRY_RECORD;
   type REQUIRED_SELECT_ENTRY is access REQUIRED_SELECT_ENTRY_RECORD;

   type REQUIRED_SELECT_ENTRY_RECORD is
      record
         ROUTINE      : ROUTINE_NAME;
         PARAMETER    : PARAMETER_TYPE;
         RESULT       : RESULT_TYPE;
         PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
         NEXT_SELECT  : REQUIRED_SELECT_ENTRY;
      end record;

   REQUIRED_SELECT_LIST : REQUIRED_SELECT_ENTRY;

procedure REQUIRED_FOR
   (ROUTINE      : ROUTINE_NAME;
    PARAMETER    : PARAMETER_TYPE;
    RESULT       : RESULT_TYPE;
    PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null) is
   TRACER : REQUIRED_SELECT_ENTRY := REQUIRED_SELECT_LIST;
   -- list is unordered.  Is there an ordering scheme I should be using???
begin
   while TRACER /= null and then
      (not (ROUTINE      = TRACER.ROUTINE and then
            PARAMETER    = TRACER.PARAMETER and then
            RESULT       = TRACER.RESULT and then
            PROGRAM_TYPE = TRACER.PROGRAM_TYPE)) loop
      TRACER := TRACER.NEXT_SELECT;
   end loop;
   if TRACER = null then
      REQUIRED_SELECT_LIST := new REQUIRED_SELECT_ENTRY_RECORD'
                                   (ROUTINE      => ROUTINE,
                                    PARAMETER    => PARAMETER,
                                    RESULT       => RESULT,
                                    PROGRAM_TYPE => PROGRAM_TYPE,
                                    NEXT_SELECT  => REQUIRED_SELECT_LIST);
      if PARAMETER = STAR then
         PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.STAR_TYPE_DECLARATION);
      end if;
      if PARAMETER = DATABASE_VALUE or RESULT = DATABASE_VALUE then
         DATABASE_TYPE.REQUIRED_FOR (PROGRAM_TYPE);
      end if;
      if PARAMETER = PROGRAM_VALUE then
         PROGRAM_CONVERSION.REQUIRED_FOR (PROGRAM_TYPE);
      end if;
   end if;
end REQUIRED_FOR;

procedure PRINT_ROUTINE_NAME
```

```
      (ROUTINE : ROUTINE_NAME) is
   begin
      case ROUTINE is
         when SELEC           => PRINT ("SELEC");
         when SELECT_ALL      => PRINT ("SELECT_ALL");
         when SELECT_DISTINCT => PRINT ("SELECT_DISTINCT");
      end case;
   end PRINT_ROUTINE_NAME;

   procedure PRINT_RESULT_TYPE
      (RESULT       : RESULT_TYPE;
       PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   begin
      case RESULT is
         when SQL_OBJECT  => PRINT ("ADA_SQL_FUNCTIONS.SQL_OBJECT");
         when INSERT_ITEM => PRINT ("ADA_SQL_FUNCTIONS.INSERT_ITEM");
         when DATABASE_VALUE =>
            PRINT ("ADA_SQL.");
            PRINT (STRING(PROGRAM_TYPE.SCHEMA_UNIT.NAME.all) & "_TYPE_PACKAGE.");
            PRINT (STRING(PROGRAM_TYPE.NAME.all) & "_TYPE");
         when PROCEDURE_CALL => null;
      end case;
   end PRINT_RESULT_TYPE;

   procedure PRINT_OPERATION
      (ROUTINE : ROUTINE_NAME) is
   begin
      case ROUTINE is
         when SELEC           => PRINT ("ADA_SQL_FUNCTIONS.O_SELEC");
         when SELECT_ALL      => PRINT ("ADA_SQL_FUNCTIONS.O_SELECT_ALL");
         when SELECT_DISTINCT => PRINT ("ADA_SQL_FUNCTIONS.O_SELECT_DISTINCT");
      end case;
   end PRINT_OPERATION;

   procedure PRINT_PARAMETER_TYPE
      (PARAMETER    : PARAMETER_TYPE;
       PROGRAM_TYPE : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR) is
   begin
      case PARAMETER is
         when STAR           => null;
         when SQL_OBJECT     => PRINT ("ADA_SQL_FUNCTIONS.SQL_OBJECT");
         when DATABASE_VALUE => PRINT_RESULT_TYPE (DATABASE_VALUE, PROGRAM_TYPE);
         when PROGRAM_VALUE  =>
            PRINT (STRING(PROGRAM_TYPE.FULL_PACKAGE_NAME.all) & ".");
            PRINT (STRING(PROGRAM_TYPE.NAME.all));
      end case;
   end PRINT_PARAMETER_TYPE;

   procedure POST_PROCESSING_1 is
```

```
   TRACER : REQUIRED_SELECT_ENTRY := REQUIRED_SELECT_LIST;
begin
   while TRACER /= null loop
      if TRACER.PARAMETER = STAR then
         SET_INDENT (2);
         if TRACER.RESULT /= PROCEDURE_CALL then
            -- Format 1: SELECT * function
            PRINT ("function ");
         else
            -- Format 2: SELECT * procedure
            PRINT ("procedure ");
         end if;
         PRINT_ROUTINE_NAME (TRACER.ROUTINE);
         PRINT_LINE;
         SET_INDENT (4);
         PRINT ("( WHAT     : STAR_TYPE;");
         PRINT_LINE;
         SET_INDENT (6);
         PRINT ("FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;");
         PRINT_LINE;
         PRINT ("WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
         PRINT_LINE;
         PRINT ("             ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;");
         PRINT_LINE;
         PRINT ("GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
         PRINT_LINE;
         PRINT ("             ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;");
         PRINT_LINE;
         PRINT ("HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
         PRINT_LINE;
         PRINT ("             ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )");
         if TRACER.RESULT /= PROCEDURE_CALL then
            -- Format 1: SELECT * function
            PRINT_LINE;
            SET_INDENT (4);
            PRINT ("return ");
            PRINT_RESULT_TYPE (TRACER.RESULT, TRACER.PROGRAM_TYPE);
         end if;
         PRINT (";");
      elsif TRACER.RESULT /= PROCEDURE_CALL then
         -- Format 3: SELECT functions other than SELECT *
         SET_INDENT (2);
         PRINT ("function ");
         PRINT_ROUTINE_NAME (TRACER.ROUTINE);
         PRINT (" is new");
         PRINT_LINE;
         SET_INDENT (4);
         PRINT ("ADA_SQL_FUNCTIONS.SELECT_LIST_SUBQUERY");
         PRINT_LINE;
```

```
            SET_INDENT (6);
            PRINT ("( ");
            PRINT_OPERATION (TRACER.ROUTINE);
            PRINT (",");
            PRINT_LINE;
            SET_INDENT (8);
            PRINT_PARAMETER_TYPE (TRACER.PARAMETER, TRACER.PROGRAM_TYPE);
            PRINT (",");
            PRINT_LINE;
            PRINT_RESULT_TYPE (TRACER.RESULT, TRACER.PROGRAM_TYPE);
            PRINT (" );");
         else
            -- Format 4: SELECT procedure other than SELECT *
            SET_INDENT (2);
            PRINT ("procedure ");
            PRINT_ROUTINE_NAME (TRACER.ROUTINE);
            PRINT (" is new");
            PRINT_LINE;
            SET_INDENT (4);
            PRINT ("ADA_SQL_FUNCTIONS.SELECT_LIST_SELECT");
            PRINT_LINE;
            SET_INDENT (6);
            PRINT ("( ");
            PRINT_OPERATION (TRACER.ROUTINE);
            PRINT (",");
            PRINT_LINE;
            SET_INDENT (8);
            PRINT_PARAMETER_TYPE (TRACER.PARAMETER, TRACER.PROGRAM_TYPE);
            PRINT (" );");
         end if;
         PRINT_LINE;
         BLANK_LINE;
         TRACER := TRACER.NEXT_SELECT;
      end loop;
   end POST_PROCESSING_1;

   procedure POST_PROCESSING_2 is
      TRACER : REQUIRED_SELECT_ENTRY := REQUIRED_SELECT_LIST;
   begin
      while TRACER /= null loop
         if TRACER.PARAMETER = STAR then
            if TRACER.RESULT /= PROCEDURE_CALL then
               -- Format 1: SELECT * function
               SET_INDENT (2);
               PRINT ("function SELEC_STAR_SUBQUERY is new");
               PRINT_LINE;
               SET_INDENT (4);
               PRINT ("ADA_SQL_FUNCTIONS.STAR_SUBQUERY");
               PRINT_LINE;
```

```
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_OPERATION (TRACER.ROUTINE);
      PRINT (", ");
      PRINT_LINE;
      SET_INDENT (8);
      PRINT_RESULT_TYPE (TRACER.RESULT, TRACER.PROGRAM_TYPE);
   else
      -- Format 2: SELECT * procedure
      SET_INDENT (2);
      PRINT ("procedure SELEC_STAR is new");
      PRINT_LINE;
      SET_INDENT (4);
      PRINT ("ADA_SQL_FUNCTIONS.STAR_SELECT");
      PRINT_LINE;
      SET_INDENT (6);
      PRINT ("( ");
      PRINT_OPERATION (TRACER.ROUTINE);
   end if;
   PRINT (" );");
   PRINT_LINE;
   BLANK_LINE;
   SET_INDENT (2);
   if TRACER.RESULT /= PROCEDURE_CALL then
      -- Format 1: SELECT * function
      PRINT ("function ");
   else
      -- Format 2: SELECT * procedure
      PRINT ("procedure ");
   end if;
   PRINT_ROUTINE_NAME (TRACER.ROUTINE);
   PRINT_LINE;
   SET_INDENT (4);
   PRINT ("( WHAT     : STAR_TYPE;");
   PRINT_LINE;
   SET_INDENT (6);
   PRINT ("FROM     : ADA_SQL_FUNCTIONS.TABLE_LIST;");
   PRINT_LINE;
   PRINT ("WHERE    : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
   PRINT_LINE;
   PRINT ("           ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;");
   PRINT_LINE;
   PRINT ("GROUP_BY : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
   PRINT_LINE;
   PRINT ("           ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT;");
   PRINT_LINE;
   PRINT ("HAVING   : ADA_SQL_FUNCTIONS.SQL_OBJECT :=");
   PRINT_LINE;
   PRINT ("           ADA_SQL_FUNCTIONS.NULL_SQL_OBJECT )");
```

```
         if TRACER.RESULT /= PROCEDURE_CALL then
            -- Format 1: SELECT * function
            PRINT_LINE;
            SET_INDENT (4);
            PRINT ("return ");
            PRINT_RESULT_TYPE (TRACER.RESULT, TRACER.PROGRAM_TYPE);
         end if;
         PRINT (" is");
         PRINT_LINE;
         SET_INDENT (2);
         PRINT ("begin");
         PRINT_LINE;
         if TRACER.RESULT /= PROCEDURE_CALL then
            -- Format 1: SELECT * function
            PRINT ("  return SELEC_STAR_SUBQUERY ");
         else
            PRINT ("  SELEC_STAR ");
         end if;
         PRINT ("( FROM, WHERE, GROUP_BY, HAVING );");
         PRINT_LINE;
         PRINT ("end ");
         PRINT_ROUTINE_NAME (TRACER.ROUTINE);
         PRINT (";");
         PRINT_LINE;
         BLANK_LINE;
      end if;
      TRACER := TRACER.NEXT_SELECT;
   end loop;
end POST_PROCESSING_2;

end SELEC;
```

3.11.62 package names.ada

```
-- names.ada -- parsing of various types of names

with CORRELATION, DDL_DEFINITIONS, ENUMERATION, FROM_CLAUSE;
package NAME is

-- This package contains the NAME.AT_CURRENT_INPUT_POINT routine, which
-- parses, by applying Ada and SQL semantics, various kinds of names.  For our
-- purposes, a "name" is a sequence of identifiers, separated by periods (if
-- there is more than one).  The following are the formats of names we
-- recognize, along with enumeration values descriptive of the kind of name:
-- (The parenthesized arguments and apostrophes are not part of the names;
-- they are shown merely to give context.  Where no explicit ending delimiter
-- is shown for the name, it is any delimiter other than a period.)

  -- table.column                              OF_QUALIFIED_COLUMN
  -- correlation.column                        OF_CORRELATED_COLUMN
```

```
-- column                                     OF_UNQUALIFIED_COLUMN
-- CONVERT_TO.package.type ( ... )            OF_CONVERT_FUNCTION
-- type ( ... )                               OF_PROGRAM_TYPE
-- type ' ( ... )                             OF_PROGRAM_TYPE
-- package.type ( ... )                       OF_PROGRAM_TYPE
-- package.type ' ( ... )                     OF_PROGRAM_TYPE
-- package.ADA_SQL.type ( ... )               OF_PROGRAM_TYPE
-- package.ADA_SQL.type ' ( ... )             OF_PROGRAM_TYPE
-- enumeration_literal                        OF_ENUMERATION_LITERAL
-- package.ADA_SQL.enumeration_literal        OF_ENUMERATION_LITERAL
-- variable                                   OF_VARIABLE
-- package.variable                           OF_VARIABLE

-- Note that we do not make a distinction between type conversions and type
-- qualifications -- they both require that the returned information designate
-- the type involved, and the calling routine can easily check the input
-- character following the name to determine which is present when a NAME.OF_-
-- PROGRAM_TYPE is seen.

-- Also note that we do not make a distinction between qualified and
-- unqualified program names.  Once the name is parsed we do not care how it
-- was designated; only the entity it denotes is important.

-- The following is the information that further processing must know about
-- each type of name parsed:
--
--   OF_QUALIFIED_COLUMN
--     table information: name, package in which table is declared, etc.
--     column information: name, type, package in which type is declared, etc.
--   OF_CORRELATED_COLUMN
--     correlation name information: name, table designated, etc.
--     column information: name, type, package in which type is declared, etc.
--   OF_UNQUALIFIED_COLUMN
--     column information: name, type, package in which type is declared, etc.
--     (The column's table is uniquely determined by SQL semantics, but need
--      not be known by our further processing once the column name is parsed,
--      which includes applying SQL semantics to determine the unique table.
--      The table can be deduced from the data structure we use, anyway, since
--      column information includes a pointer to table information.)
--   OF_CONVERT_FUNCTION
--     type information: name, package in which declared, etc.
--   OF_PROGRAM_TYPE
--     type information: name, package in which declared, etc.
--   OF_ENUMERATION_LITERAL
--     list of possible types of which the (potentially) overloaded
--      enumeration literal may be a value (see enums.ada)
--   OF_VARIABLE
--     type information: name, package in which declared, etc.
```

```
-- In all cases, information about a type includes the class of the type and
-- all information required to enforce strong typing and/or generate
-- conversion functions, as described in pgmconvs.ada.

-- The following data structure contains all the information that must be
-- known about a name:

  type KIND is
  ( OF_QUALIFIED_COLUMN , OF_CORRELATED_COLUMN , OF_UNQUALIFIED_COLUMN ,
    OF_CONVERT_FUNCTION , OF_PROGRAM_TYPE      , OF_ENUMERATION_LITERAL ,
    OF_VARIABLE );

  type INFORMATION ( KIND : NAME.KIND := OF_QUALIFIED_COLUMN ) is
    record
      NUMBER_OF_TOKENS          : POSITIVE;
      case KIND is
        when OF_QUALIFIED_COLUMN =>
          TABLE                 : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
          QUALIFIED_COLUMN      : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when OF_CORRELATED_COLUMN =>
          CORRELATION_NAME      : CORRELATION.NAME_DECLARED_ENTRY;
          CORRELATED_COLUMN     : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when OF_UNQUALIFIED_COLUMN =>
          UNQUALIFIED_COLUMN    : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when OF_CONVERT_FUNCTION =>
          CONVERT_TO_TYPE       : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when OF_PROGRAM_TYPE =>
          PROGRAM_TYPE          : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when OF_ENUMERATION_LITERAL =>
          ENUMERATION_TYPE_LIST : ENUMERATION.TYPE_LIST;
        when OF_VARIABLE =>
          VARIABLE_TYPE         : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
      end case;
    end record;

-- Note the NUMBER_OF_TOKENS component: When NAME.AT_CURRENT_INPUT_POINT is
-- called, the lexical analyzer state is such that FIRST_LOOK_AHEAD_TOKEN will
-- return the first identifier in the name.  Return is made in the same state
-- (i.e., tokens are not gobbled), with the lexical look ahead pointer reset
-- such that both NEXT_LOOK_AHEAD_TOKEN and FIRST_LOOK_AHEAD_TOKEN will return
-- the first identifier in the name.  This is done to facilitate the calling
-- routine pointing to an error in the name if necessary.  NUMBER_OF_TOKENS
-- tells how many tokens the calling routine must gobble to skip over the name
-- parsed from the input stream.

-- Before calling NAME.AT_CURRENT_INPUT_POINT, the calling routine verifies
-- that the next look ahead token is indeed an identifier, and that it is not
-- a reserved word (e.g., AVG), so that it is definitely the start of a name.
```

```
-- Various restrictions can be placed on the names to be recognized (see
-- below).  In the event an error is detected in a name, or a name does not
-- satisfy the restrictions given, the SYNTAX_ERROR exception is raised so
-- that processing can automatically be continued at the next synchronization
-- point (typically the end of the statement being processed).

-- Restriction 1: Kind of name recognized.  We can restrict the kind of name
-- recognized, to one of the following categories:
--
--   NAME.IS_ANYTHING - any kind of name will be recognized
--
--   NAME.IS_COLUMN_SPECIFICATION - only column specifications (column name
--    with optional qualifying table name or correlation name) will be
--    recognized.  See type conversion restriction for special note on also
--    recognizing CONVERT_TO.
--
--   NAME.IS_COLUMN_NAME - only column names (no optional qualifying table
--    name or correlation name) will be recognized.
--
--   NAME.IS_PROGRAM_VALUE - only program values (no database columns or
--    CONVERT_TOs) will be recognized
--
--   NAME.IS_PROGRAM_VARIABLE - only program variables will be recognized.
--    See type conversion restriction for special note on also recognizing Ada
--    type conversions.

  type KIND_RESTRICTION is
  ( IS_ANYTHING      , IS_COLUMN_SPECIFICATION , IS_COLUMN_NAME ,
    IS_PROGRAM_VALUE , IS_PROGRAM_VARIABLE );

-- Restriction 2: Scope.  Column specifications and column names are
-- recognized within the context of their scope - the (possibly nested) from
-- clauses that apply to the point in the statement at which the name is being
-- processed.  The applicable scope is indicated to NAME.AT_CURRENT_INPUT_-
-- POINT by a parameter of type FROM_CLAUSE.INFORMATION (see froms.ada).  For
-- calls where scope is not applicable (NAME.IS_PROGRAM_VALUE or NAME.IS_-
-- PROGRAM_VARIABLE) the scope parameter is NULL.  A BOOLEAN flag parameter,
-- THIS_SCOPE_ONLY, causes a name to be recognized based on only the innermost
-- from clause if TRUE, or based on all nested from clauses if FALSE.

-- Restriction 3: Type conversion.  A BOOLEAN flag, ALLOW_TYPE_CONVERSION, is
-- applicable to the NAME.IS_COLUMN_SPECIFICATION and NAME.IS_PROGRAM_VARIABLE
-- restrictions.
--
-- (1) There are contexts where SQL requires a column specification, rather
--     than a general value expression.  It is the purpose of the NAME.IS_-
--     COLUMN_SPECIFICATION name restriction to ensure that we do indeed have
--     a column specification, rather than a program value.  With our strong
--     typing, however, it may make sense to apply a CONVERT_TO function to a
```

```
--     column specification.  If ALLOW_TYPE_CONVERSION is TRUE with the
--     restriction NAME.IS_COLUMN_SPECIFICATION, then a NAME.OF_CONVERT_-
--     FUNCTION will be recognized.  The calling routine will then repeat its
--     call to obtain the actual column specification (or indication of
--     another CONVERT_TO function, since these things could theoretically,
--     though pathologically, be nested without limit).  If ALLOW_TYPE_-
--     CONVERSION is FALSE with the restriction NAME.IS_COLUMN_SPECIFICATION,
--     then a NAME.OF_CONVERT_FUNCTION will not be recognized.
--
-- (2) The NAME.IS_PROGRAM_VARIABLE restriction is used to recognize cursor
--     names and parameters to INTO routines.  A cursor name must stand by
--     itself in an Ada/SQL statement; no conversions or expressions are
--     possible.  Parameters to INTO routines are Ada OUT, and so must be
--     variables or type conversions.  To recognize an INTO parameter, NAME.-
--     AT_CURRENT_INPUT_POINT is called with the NAME.IS_PROGRAM_VARIABLE
--     restriction and ALLOW_TYPE_CONVERSION => TRUE.  Either a variable name
--     or an Ada type conversion will be recognized.  If a type conversion was
--     found, the calling routine will then again call NAME.AT_CURRENT_INPUT_-
--     POINT with the NAME.IS_PROGRAM_VARIABLE restriction, except that this
--     time ALLOW_TYPE_CONVERSION will be set to FALSE.  This call will
--     recognize only a program a variable, which is required since type
--     conversions may not be nested on OUT parameters.

-- Only the following restriction combinations are permitted by NAME.AT_-
-- CURRENT_INPUT_POINT.  Examples of how they are used in processing Ada/SQL
-- statements follow, keyed to the numbers in the table.  "n/a" means that
-- the indicated parameter is logically not applicable to the restriction, but
-- must be set as shown anyway.
--
--                                  THIS_SCOPE_-     ALLOW_TYPE_-
-- Kind of name restriction         ONLY             CONVERSION       Note
-- ------------------------         ------------     ------------     ----
-- NAME.IS_ANYTHING                 TRUE             TRUE             (1)
-- NAME.IS_ANYTHING                 FALSE            TRUE             (2)
-- NAME.IS_COLUMN_SPECIFICATION     TRUE             FALSE            (3)
-- NAME.IS_COLUMN_SPECIFICATION     FALSE            TRUE             (4)
-- NAME.IS_COLUMN_SPECIFICATION     FALSE            FALSE            (5)
-- NAME.IS_COLUMN_NAME              TRUE             FALSE            (6)
-- NAME.IS_PROGRAM_VALUE            TRUE (n/a)       TRUE             (7)
-- NAME.IS_PROGRAM_VARIABLE         TRUE (n/a)       TRUE             (8)
-- NAME.IS_PROGRAM_VARIABLE         TRUE (n/a)       FALSE            (9)
--
-- (1) Called when processing a primary in a context that does not permit
--     outer references (e.g., the result specification of a subquery) or
--     where an outer reference would not make sense (e.g., value expression
--     in the set clause of an update statement)
--
-- (2) Called when processing a primary in a context that permits outer
--     references (most contexts)
```

```
--
-- (3) Called for processing group by and order by clauses
--
-- (4) Called when processing a distinct set function or a like predicate
--
-- (5) Called when processing a null predicate
--
-- (6) Called when processing various DDL statements (not yet implemented),
--     the insert column list in an insert statement, and each object column
--     within an update statement
--
-- (7) Called when processing the parameter to an INDICATOR function, any
--     value specification (the calling routine recognizes and gobbles
--     INDICATOR; we do not), and the argument to an Ada type conversion or
--     qualification
--
-- (8) First call for each parameter to INTO routine, as described above
--
-- (9) Second call (if required) for each parameter to INTO routine, as
--     described above; also called to process cursor name

-- Here's the routine (finally!):

  function AT_CURRENT_INPUT_POINT
            ( SCOPE                  : FROM_CLAUSE.INFORMATION;
              RESTRICT_SO            : KIND_RESTRICTION;
              THIS_SCOPE_ONLY        : BOOLEAN;
              ALLOW_TYPE_CONVERSION  : BOOLEAN;
              REPORT_ERRORS          : BOOLEAN := TRUE ) return INFORMATION;

-- Here is a sketch of some of the processing performed by NAME.AT_CURRENT_-
-- INPUT_POINT:

-- General error check:

-- If any item from column A is seen, it must be verified to not be the same as
-- any item in column B.

-- Column A                                 Column B

-- table name in table.column               type name from a used package
-- column (unqualified)                     package name (any DDL package)
-- enumeration literal used unqualified     correlation name
--                                          variable name from a used package
---------------------------------------------------------------------------------
-- We may define restrictions on what are legal names at a particular point
-- (report error on violation):
--  Ada or Ada/SQL context requires program value
--  SQL context requires column specification
```

```
--  SQL context requires column name
--  Ada/SQL context requires program variable

-- We may also define whether or not type conversions are permitted (option
-- allowed only on column specification and program variable)

-- The following are program values:
--
-- type ( ... )
-- type ' ( ... )
-- package.type ( ... )
-- package.type ' ( ... )
-- package.ADA_SQL.type ( ... )
-- package.ADA_SQL.type ' ( ... )
-- enumeration_literal
-- package.ADA_SQL.enumeration_literal
-- variable
-- package.variable

-- The following are column specifications:
--
-- table.column
-- correlation.column
-- column
-- CONVERT_TO.package.type ( ... ) if type conversion is allowed

-- The following is a column name:
--
-- column

-- The following are program variables:
--
-- variable
-- variable.package
-- type ( ... ) if type conversion is allowed
-- package.type ( ... ) if type conversion is allowed
-- package.ADA_SQL.type ( ... ) if type conversion is allowed
------------------------------------------------------------------------------
-- We may also restrict how far back we look through nested selects:
--  Look only at this scope
--  Look at any scope
------------------------------------------------------------------------------
-- Discussion of one part names:

-- column
--   must be declared in exactly one table at appropriate scope (innermost if
--    looking only at this scope, innermost scope in which it is declared if
--    looking at any scope)
-- type
```

```
--   must be declared in exactly one used library unit (may still get error on
--    compile if nested ADA_SQL package had not been used - checking that is
--    beyond our current scope)
--   cannot be same as any DDL package name or correlation name
--   must be integer, enumeration, floating point, or string (not record)
-- enumeration_literal
--   must be declared in at least one used library unit (multiple declarations
--    possible) (may still get error on compile if nested ADA_SQL package had
--    not been used - checking that is beyond our current scope)
-- variable
--   must be declared in exactly one used package
--   cannot be same as any DDL package name or correlation name

-- Error if a name cannot be established to be exactly one of the above,
-- except:
--  If Ada context requires program value, then may be both a column and an
--   enumeration_literal, return enumeration_literal
--  If SQL context requires column specification or column name, then may be
--   both a column and an enumeration_literal, return column
------------------------------------------------------------------------------
-- Discussion of two part names

-- table.column
--   table must be named in from clause at this or any scope, depending on
--    restriction
--   column must be present in table
-- correlation.column
--   correlation name must be named in from clause at this or any scope,
--    depending on restriction
--   column must be present in table designated by correlation name
-- package.variable
--   package must be with'ed
--   variable must be declared in package
--   error if package name is also a correlation name
-- package.type
--   package (predefined, e.g., DATABASE) must be with'ed or else be STANDARD
--   type must be declared directly in package
--   type must be integer, enumeration, floating point, or string
--   error if package name is also a correlation name

-- Error if a name cannot be established to be exactly one of the above
-----------------------------------------------------------------------------
-- Discussion of three part names

-- CONVERT_TO.package.type
--   CONVERT_TO is considered a reserved word -- if we see CONVERT_TO, we have
--    one of these things
--   package must be with'ed, unless STANDARD
--   type must be declared in package.ADA_SQL (or package if predefined, e.g.,
```

```
--    DATABASE)
--   type must be integer, enumeration, floating point, or string (not record)
--   error if type and enumeration literal with same name is declared in
--    package.ADA_SQL, or package if predefined (this should be a DDL reader
--    error, but is not; there should be no errors in predefined packages)
-- package.ADA_SQL.type
--   package must be with'ed
--   type must be declared in package.ADA_SQL
--   type must be integer, enumeration, floating point, or string (not record)
--   error if type and enumeration literal with same name is declared in
--    package.ADA_SQL (this should be a DDL reader error, but is not)
-- package.ADA_SQL.enumeration_literal
--   package must be with'ed
--   enumeration_literal must be declared at least once in package.ADA_SQL
--    (multiple declarations possible)
--   error if type and enumeration literal with same name is declared in
--    package.ADA_SQL (this should be a DDL reader error, but is not)

-- Error if package name is also a correlation name

  function IS_PACKAGE_WITHED
          (PAK : STRING)
           return BOOLEAN;
end NAME;
```

3.11.63 package nameb.ada

```
with LEXICAL_ANALYZER, DDL_DEFINITIONS, EXTRA_DEFINITIONS, DDL_VARIABLES;
with CORRELATION;
use DDL_DEFINITIONS, CORRELATION, LEXICAL_ANALYZER;

-- names.ada -- parsing of various types of names
--
-- details on the processing of the different possible names to recognize
--
-- information.number_of_tokens gets updated when name parts are read
-- and when we're done set next_look_ahead_token and first_look_ahead_token and
-- next_look_ahead_token are the same
--
--           names
-- part_1       part_2    part_3         how to process
--
-- table        column    none  of_qualified_column
--                              returns in information ddl full_name_descriptor
--                                                       for table & column
--                              consider parm_this_scope_only
--                              part_1 (table name) must not be same as any
--                                    type name from any used package
--                              part_1 (table name) must be named in from
--                                    clause in appropriate scope
```

```
--                                      part_2 (column name) must be in table (part_1)
--                                      parm_restrict_so = is_anything - ok
--                                      parm_restrict_so = is_column_specification - ok
--                                      parm_restrict_so = is_column_name - no
--                                      parm_restrict_so = is_program_value - no
--                                      parm_restrict_so = is_program_variable - no
-- correlation  column     none         of_correlated_column
--                                      returns in information name_declared_entry for
--                                                             correlation name and
--                                                             ddl full_name_descriptor
--                                                             for column
--                                      consider parm_this_scope_only
--                                      part_1 (correlation name) must be named in
--                                            appropriate scope
--                                      part_2 (column name) must be in table
--                                            designated by correlation name (part_1)
--                                      parm_restrict_so = is_anything - ok
--                                      parm_restrict_so = is_column_specification - ok
--                                      parm_restrict_so = is_column_name - no
--                                      parm_restrict_so = is_program_value - no
--                                      parm_restrict_so = is_program_variable - no
-- column       none       none         of_unqualified_column
--                                      returns in information ddl full_name_descriptor
--                                                             for column
--                                      consider parm_this_scope_only
--                                      part_1 (column name) must not be the same as
--                                            any ddl package name
--                                      part_1 (column name) must be declared in only
--                                            one table at appropriate scope
--                                      parm_restrict_so = is_anything - ok
--                                      parm_restrict_so = is_column_specification - ok
--                                      parm_restrict_so = is_column_name - ok
--                                      parm_restrict_so = is_program_value - no
--                                      parm_restrict_so = is_program_variable - no
-- convert_to   package    type         of_convert_function
--                                      returns in information ddl full_name_descriptor
--                                                             for the type
--                                      part_2 (package name) must be withed
--                                      part_2 (package name) may have a subpackage
--                                            ADA_SQL but it must not be spelled out
--                                            here
--                                      part_2 (type name) must not be the same as any
--                                            and enumeration literal in the same
--                                            package (part_2)
--                                      part_2 (package name) must not be the same as
--                                            any correlation name
--                                      part_3 (type name) must be declared in the
--                                            package (part_2) or the package.ada_sql
--                                      part_3 (type) must be integer, enumeration,
```

```
--                                             float or string
--                                       part_3 (type) must not be record
--                                       parm_restrict_so = is_anything - ok
--                                       parm_restrict_so = is_column_specification
--                                              if allow_type_conversion = true - yes
--                                              if allow_type_conversion = false - no
--                                       parm_restrict_so = is_column_name - no
--                                       parm_restrict_so = is_program_value - no
--                                       parm_restrict_so = is_program_variable - no
-- type          none       none        of_program_type
--                                       returns in information ddl full_name_descriptor
--                                                              for the type
--                                       part_1 (type name) must be declared in exactly
--                                             one used package
--                                       part_1 (type name) must not be same as and ddl
--                                             package name
--                                       part_1 (type name) must not be same as any
--                                             correlation name
--                                       part_1 (type) must be integer, enumeration,
--                                             float, string
--                                       part_1 (type) must not be record
--                                       parm_restrict_so = is_anything - ok
--                                       parm_restrict_so = is_column_specification - no
--                                       parm_restrict_so = is_column_name - no
--                                       parm_restrict_so = is_program_value - yes
--                                       parm_restrict_so = is_program_variable
--                                              if allow_type_conversion = true - yes if
--                                                        not type qualification (')
--                                              if allow_type_conversion = false - no
-- package       type       none        of_program_type
--                                       returns in information ddl full_name_descriptor
--                                                              for the type
--                                       package (part_1) must be a package without
--                                             ADA_SQL subpackage
--                                       part_1 (package name) must not be the same as
--                                             any correlation name
--                                       part_2 (type) must be declared in the package
--                                             (part_1)
--                                       part_2 (type) must be integer, enumeration,
--                                             float or string
--                                       parm_restrict_so = is_anything - ok
--                                       parm_restrict_so = is_column_specification - no
--                                       parm_restrict_so = is_column_name - no
--                                       parm_restrict_so = is_program_value - yes
--                                       parm_restrict_so = is_program_variable
--                                              if allow_type_conversion = true - yes if
--                                                        not type qualification (')
--                                              if allow_type_conversion = false - no
--                                       package name must be withed
```

```
-- package      ada_sql   type  of_program_type
--                              returns in information ddl full_name_descriptor
--                                                      for the type
--                              part_1 (package name) must not be the same as
--                                    any correlation name
--                              part_3 (type name) must be declared in package
--                                    (part_1.part_2)
--                              part_3 (type) must be integer, enumeration,
--                                    float or string
--                              part_3 (type) must not be record
--                              part_3 (type name) must not be the same as any
--                                    enumeration literal the package
--                                    (part_1.part_2)
--                              parm_restrict_so = is_anything - ok
--                              parm_restrict_so = is_column_specification - no
--                              parm_restrict_so = is_column_name - no
--                              parm_restrict_so = is_program_value - yes
--                              parm_restrict_so = is_program_variable
--                                     if allow_type_conversion = true - yes if
--                                               not type qualification (')
--                                     if allow_type_conversion = false - no
--                              package name must be withed
-- enum         none      none  of_enumeration_literal
--                              returns in information a type_list of all
--                                                      possible types that the
--                                                      enumeration literal
--                                                      could indicate in
--                                                      this case
--                              part_1 (enumeration literal) must not be the
--                                    same as any correlation name
--                              part_1 (enumeration literal) must not be the
--                                    same as any variable name from any used
--                                    package
--                              part_1 (enumeration literal) must be declared
--                                    in at least one used package
--                              parm_restrict_so = is_anything - ok
--                              parm_restrict_so = is_column_specification - no
--                              parm_restrict_so = is_column_name - no
--                              parm_restrict_so = is_program_value - yes
--                              parm_restrict_so = is_program_variable - no
-- package      ada_sql   enum  of_enumeration_literal
--                              returns in information a type_list of all
--                                                      possible types that the
--                                                      enumeration literal
--                                                      could indicate in
--                                                      this case
--                              part_1 (package name) must be withed
--                              part_1 (package name) must not be the same as
--                                    any correlation name
```

```
--                                        part_3 (enumeration literal) must be declared
--                                              at least once in the package
--                                              (part_1.part_2)
--                                        part_3 (enumeration literal) must not be the
--                                              same as any type name in the package
--                                              (part_1.part_2)
--                                        parm_restrict_so = is_anything - ok
--                                        parm_restrict_so = is_column_specification - no
--                                        parm_restrict_so = is_column_name - no
--                                        parm_restrict_so = is_program_value - yes
--                                        parm_restrict_so = is_program_variable - no
-- variable     none      none   of_variable
--                                        returns in information ddl full_name_descriptor
--                                                                for the variable
--                                        part_1 (variable) must be declared in exactly
--                                              one used package
--                                        part_1 (variable) must not be the same as any
--                                              ddl package name
--                                        part_1 (variable) must not be the same as any
--                                              correlation name
--                                        parm_restrict_so = is_anything - ok
--                                        parm_restrict_so = is_column_specification - no
--                                        parm_restrict_so = is_column_name - no
--                                        parm_restrict_so = is_program_value - yes
--                                        parm_restrict_so = is_program_variable - yes
-- package      variable  none   of_variable
--                                        returns in information ddl full_name_descriptor
--                                                                for the variable
--                                        part_1 (package name) must be withed
--                                        part_1 (package name) must not be same as any
--                                              correlation name
--                                        part_2 (variable) must be declared in the
--                                              package (part_1)
--                                        parm_restrict_so = is_anything - ok
--                                        parm_restrict_so = is_column_specification - no
--                                        parm_restrict_so = is_column_name - no
--                                        parm_restrict_so = is_program_value - yes
--                                        parm_restrict_so = is_program_variable - yes
------------------------------------------------------------------------
-- if one part name
--    if restrict_so = is_program_value and the name is both a column and
--                     an enumeration literal - use the enumeration literal
--    if restrict_so = is_column_specification or is_column_name and the name
--                     is both a column and an enumeration literal - use column
------------------------------------------------------------------------
--
-- if parts_count = 1 then
--   if parm_restrict_so = is_program_value then
--     if    enum      none     none  of_enumeration_literal  then return true
```

```
--     end if
--   end if
--   if      column      none     none   of_unqualified_column   then return true
--   elsif   enum        none     none   of_enumeration_literal then return true
--   elsif   type        none     none   of_program_type         then return true
--   elsif   variable    none     none   of_variable             then return true
--   end if
-- elsif parts_count = 2 then
--   if      correlation column   none   of_correlated_column    then return true
--   elsif   package     type     none   of_program_type         then return true
--   elsif   package     variable none   of_variable             then return true
--   elsif   table       column   none   of_qualified_column     then return true
--   end if
-- elsif parts_count = 3 then
--   if      convert_to  package  type   of_convert_function     then return true
--   elsif   package     ada_sql  enum   of_enumeration_literal then return true
--   elsif   package     ada_sql  type   of_program_type         then return true
--   end if
-- end if
-- syntax error - unrecognized name


package body NAME is

  type A_S is access STRING;
  STRING_ADA_SQL                      : A_S := new STRING'("ADA_SQL");
  STRING_NULL                         : A_S := new STRING'("");
  TOKEN                               : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  INFORMATION_OF_QUALIFIED_COLUMN     : INFORMATION (OF_QUALIFIED_COLUMN);
  INFORMATION_OF_CORRELATED_COLUMN    : INFORMATION (OF_CORRELATED_COLUMN);
  INFORMATION_OF_UNQUALIFIED_COLUMN   : INFORMATION (OF_UNQUALIFIED_COLUMN);
  INFORMATION_OF_CONVERT_FUNCTION     : INFORMATION (OF_CONVERT_FUNCTION);
  INFORMATION_OF_PROGRAM_TYPE         : INFORMATION (OF_PROGRAM_TYPE);
  INFORMATION_OF_ENUMERATION_LITERAL : INFORMATION (OF_ENUMERATION_LITERAL);
  INFORMATION_OF_VARIABLE             : INFORMATION (OF_VARIABLE);
  WHAT_KIND                           : KIND;
  PARM_SCOPE                          : FROM_CLAUSE.INFORMATION;
  PARM_RESTRICT_SO                    : KIND_RESTRICTION;
  PARM_THIS_SCOPE_ONLY                : BOOLEAN;
  PARM_ALLOW_TYPE_CONVERSION          : BOOLEAN;
  PARM_REPORT_ERRORS                  : BOOLEAN;
  TYPE_QUALIFICATION                  : BOOLEAN;
  DUPLICATED                          : BOOLEAN;
  GLOBAL_FULL_NAME_DES                :
                                    DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
  GLOBAL_ENUM_TYPE_LIST               : ENUMERATION.TYPE_LIST;
  GLOBAL_CORRELATION_ENTRY            : CORRELATION.NAME_DECLARED_ENTRY;
```

```
-------------------------------------------------------------------
-- NAME_ERROR

  procedure NAME_ERROR
           (MESSAGE : STRING) is
  begin
    if PARM_REPORT_ERRORS then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, MESSAGE);
    else
      raise SYNTAX_ERROR;
    end if;
  end NAME_ERROR;

-----------------------------------------------------------------------------
-- GET_NAME_PARTS - return a count and up to three name parts, more than
--                  three is an error.  Set type_qualification to true
--                  if the token after all possible names is '
-- set globals:
--     none

  procedure GET_NAME_PARTS
           (PARTS_COUNT : in out NATURAL;
            PART_1      :    out STRING;
            PART_1_LEN  : in out NATURAL;
            PART_2      :    out STRING;
            PART_2_LEN  : in out NATURAL;
            PART_3      :    out STRING;
            PART_3_LEN  : in out NATURAL;
            TYPE_QUAL   :    out BOOLEAN;
            TOKEN_COUNT :    out POSITIVE) is

    FIRST : BOOLEAN := TRUE;
    TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
    TC    : NATURAL := 0;

  begin
    PARTS_COUNT := 0;
    PART_1_LEN := 0;
    PART_2_LEN := 0;
    PART_3_LEN := 0;
    TYPE_QUAL := FALSE;

I_LOOP:
    for I in 1..3 loop
      for J in 1..2 loop
        if FIRST then
          FIRST := FALSE;
          TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        else
```

```
    TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
  end if;
  case TOKEN.KIND is
    when LEXICAL_ANALYZER.IDENTIFIER =>
                 if J = 1 then
                   PARTS_COUNT := PARTS_COUNT + 1;
                   TC := TC + 1;
                   if I = 1 then
                     PART_1_LEN := TOKEN.ID.all'LENGTH;
                     PART_1 (1..PART_1_LEN) := TOKEN.ID.all;
                   elsif I = 2 then
                     PART_2_LEN := TOKEN.ID.all'LENGTH;
                     PART_2 (1..PART_2_LEN) := TOKEN.ID.all;
                   elsif I = 3 then
                     PART_3_LEN := TOKEN.ID.all'LENGTH;
                     PART_3 (1..PART_3_LEN) := TOKEN.ID.all;
                   end if;
                 else
                   exit I_LOOP;
                 end if;
    when LEXICAL_ANALYZER.CHARACTER_LITERAL =>
                 if J = 1 then
                   PARTS_COUNT := PARTS_COUNT + 1;
                   TC := TC + 1;
                   if I = 1 then
                     PART_1_LEN := 3;
                     PART_1 (1..PART_1_LEN) := "'" &
                             TOKEN.CHARACTER_VALUE & "'";
                   elsif I = 2 then
                     PART_2_LEN := 3;
                     PART_2 (1..PART_2_LEN) := "'" &
                             TOKEN.CHARACTER_VALUE & "'";
                   elsif I = 3 then
                     PART_3_LEN := 3;
                     PART_3 (1..PART_3_LEN) := "'" &
                             TOKEN.CHARACTER_VALUE & "'";
                   end if;
                 else
                   exit I_LOOP;
                 end if;
    when LEXICAL_ANALYZER.NUMERIC_LITERAL   => exit I_LOOP;
    when LEXICAL_ANALYZER.STRING_LITERAL    => exit I_LOOP;
    when LEXICAL_ANALYZER.DELIMITER         =>
                 if J = 2 and TOKEN.DELIMITER = LEXICAL_ANALYZER.DOT then
                   TC := TC + 1;
                   null;
                 else
                   exit I_LOOP;
                 end if;
```

```
          when LEXICAL_ANALYZER.RESERVED_WORD     => exit I_LOOP;
          when LEXICAL_ANALYZER.END_OF_FILE       => exit I_LOOP;
        end case;
      end loop;
    end loop I_LOOP;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.APOSTROPHE then
      TYPE_QUAL := TRUE;
    end if;
    LEXICAL_ANALYZER.SET_LOOK_AHEAD;
    if TC > 0 then
      TOKEN_COUNT := TC;
    end if;
  end GET_NAME_PARTS;

--------------------------------------------------------------------
-- BUILD_FULL_PACKAGE - given two possible packages, build a qualified
--                          package name
-- set globals:
-- none

  procedure BUILD_FULL_PACKAGE
           (PAK1     : STRING;
            PAK2     : STRING;
            PAK_NAME : in out STRING;
            PAK_LEN  : out NATURAL) is
     LOCAL_PAK_LEN : NATURAL := 0;
  begin
    if PAK1'LENGTH > 0 then
      LOCAL_PAK_LEN := PAK1'LENGTH;
      PAK_NAME (1..LOCAL_PAK_LEN) := PAK1;
    end if;
    if PAK2'LENGTH > 0 then
      LOCAL_PAK_LEN := LOCAL_PAK_LEN + 1;
      PAK_NAME (LOCAL_PAK_LEN) := '.';
      PAK_NAME (LOCAL_PAK_LEN + 1 .. LOCAL_PAK_LEN + PAK2'LENGTH) := PAK2;
      LOCAL_PAK_LEN := LOCAL_PAK_LEN + PAK2'LENGTH;
    end if;
    PAK_LEN := LOCAL_PAK_LEN;
  end BUILD_FULL_PACKAGE;

---------------------------------------------------------------------------
--  ANY_CORRELATION_NAME - if the name matches that of any correlation name
--                           return true, else return false
-- set globals:
-- none

  function ANY_CORRELATION_NAME
          (CORR_NAME : STRING)
```

```
         return      BOOLEAN is
  begin
    return CORRELATION.NAME_IS_DECLARED (CORR_NAME);
  end ANY_CORRELATION_NAME;

----------------------------------------------------------------------
-- ANY_TYPE_FROM_USED_PACKAGES - if the name matches that of any type from
--                               any used package return true else return
--                               false
-- set globals:
-- none

  function ANY_TYPE_FROM_USED_PACKAGES
          (TYPE_NAME : STRING)
           return      BOOLEAN is

    USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR :=
           EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
    TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
           DDL_VARIABLES.FIRST_TYPE;

  begin
    while TYPE_DES /= null loop
      if (TYPE_DES.TYPE_KIND = A_TYPE or
          TYPE_DES.TYPE_KIND = A_SUBTYPE or
          TYPE_DES.TYPE_KIND = A_DERIVED) and
          TYPE_NAME = STRING (TYPE_DES.FULL_NAME.NAME.all) then
        USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
        while USED /= null loop
          if USED.NAME.all = TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all then
            return TRUE;
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      TYPE_DES := TYPE_DES.NEXT_TYPE;
    end loop;
    return FALSE;
  end ANY_TYPE_FROM_USED_PACKAGES;

-------------------------------------------------------------------------
-- ANY_VARIABLE_FROM_USED_PACKAGES - if the name matches that of any variable
--                                   from any used package return true else
--                                   return false
-- set globals:
-- none

  function ANY_VARIABLE_FROM_USED_PACKAGES
          (VAR_NAME : STRING)
```

```
          return     BOOLEAN is

    USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR :=
           EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
    VAR_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
           DDL_VARIABLES.FIRST_VARIABLE;

  begin
    while VAR_DES /= null loop
      if VAR_DES.TYPE_KIND = A_VARIABLE and
         VAR_NAME = STRING (VAR_DES.FULL_NAME.NAME.all) then
        USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
        while USED /= null loop
          if USED.NAME.all = VAR_DES.FULL_NAME.FULL_PACKAGE_NAME.all then
            return TRUE;
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      VAR_DES := VAR_DES.NEXT_TYPE;
    end loop;
    return FALSE;
  end ANY_VARIABLE_FROM_USED_PACKAGES;

---------------------------------------------------------------------------
-- ANY_TYPE_FROM_THIS_PACKAGE - if the name matches that of any type from
--                              the package pack1.pack2 return true else
--                              return false
-- set globals:
-- none

  function ANY_TYPE_FROM_THIS_PACKAGE
          (TYPE_NAME : STRING;
           PAK1      : STRING;
           PAK2      : STRING)
           return      BOOLEAN is

    TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
           DDL_VARIABLES.FIRST_TYPE;
    PAK_NAME : STRING (1..250);
    PAK_LEN  : NATURAL;

  begin
    BUILD_FULL_PACKAGE (PAK1, PAK2, PAK_NAME, PAK_LEN);
    while TYPE_DES /= null loop
      if (TYPE_DES.TYPE_KIND = A_TYPE or
          TYPE_DES.TYPE_KIND = A_SUBTYPE or
          TYPE_DES.TYPE_KIND = A_DERIVED) and
          TYPE_NAME = STRING (TYPE_DES.FULL_NAME.NAME.all) and
```

```
        PAK_NAME(1..PAK_LEN) = STRING
                      (TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all) then
          return TRUE;
      end if;
      TYPE_DES := TYPE_DES.NEXT_TYPE;
    end loop;
    return FALSE;
  end ANY_TYPE_FROM_THIS_PACKAGE;

------------------------------------------------------------------------
-- ANY_ENUM_LIT_FROM_THIS_PACKAGE - if the name matches that of any
--                                  enumeration literal from the package
--                                  pack1.pack2 return true, else return
--                                  false
-- set globals:
-- none

  function ANY_ENUM_LIT_FROM_THIS_PACKAGE
          (ENUM_LIT : STRING;
           PAK1     : STRING;
           PAK2     : STRING)
           return     BOOLEAN is

    PAK_NAME : STRING (1..250);
    PAK_LEN  : NATURAL;
    ENUM_DES : DDL_DEFINITIONS.ACCESS_ENUM_LIT_DESCRIPTOR :=
               DDL_VARIABLES.FIRST_ENUM_LIT;
    FULL_ENUM : DDL_DEFINITIONS.ACCESS_FULL_ENUM_LIT_DESCRIPTOR;

  begin
    BUILD_FULL_PACKAGE (PAK1, PAK2, PAK_NAME, PAK_LEN);
    while ENUM_DES /= null loop
      if ENUM_LIT = STRING (ENUM_DES.NAME.all) then
        FULL_ENUM := ENUM_DES.FIRST_FULL_ENUM_LIT;
        while FULL_ENUM /= null loop
          if PAK_NAME (1..PAK_LEN) = STRING
                    (FULL_ENUM.TYPE_IS.FULL_NAME.FULL_PACKAGE_NAME.all) then
            return TRUE;
          end if;
          FULL_ENUM := FULL_ENUM.NEXT_LIT;
        end loop;
      end if;
      ENUM_DES := ENUM_DES.NEXT_ENUM_LIT;
    end loop;
    return FALSE;
  end ANY_ENUM_LIT_FROM_THIS_PACKAGE;

-------------------------------------------------------------------------
-- IS_TYPE_IN_THIS_PACKAGE - if the type name is declared in the package
```

```
--                              pack1.pack2 return true else return false
-- set globals:
-- GLOBAL_FULL_NAME_DES - full name descriptor of type

  function IS_TYPE_IN_THIS_PACKAGE
          (TYP    : STRING;
           PAK1   : STRING;
           PAK2   : STRING)
           return BOOLEAN is

    TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
           DDL_VARIABLES.FIRST_TYPE;
    PAK_NAME : STRING (1..250);
    PAK_LEN  : NATURAL;

  begin
    GLOBAL_FULL_NAME_DES := null;
    BUILD_FULL_PACKAGE (PAK1, PAK2, PAK_NAME, PAK_LEN);
    while TYPE_DES /= null loop
      if (TYPE_DES.TYPE_KIND = A_TYPE or
          TYPE_DES.TYPE_KIND = A_SUBTYPE or
          TYPE_DES.TYPE_KIND = A_DERIVED) and
          TYP = STRING (TYPE_DES.FULL_NAME.NAME.all) and
          PAK_NAME(1..PAK_LEN) = STRING
                     (TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all) then
            GLOBAL_FULL_NAME_DES := TYPE_DES.FULL_NAME;
            return TRUE;
      end if;
      TYPE_DES := TYPE_DES.NEXT_TYPE;
    end loop;
    return FALSE;
  end IS_TYPE_IN_THIS_PACKAGE;

------------------------------------------------------------------------
-- IS_TYPE_IN_ONE_USED_PACKAGE - if the type name is declared in exactly one
--                                used package return true else return false
-- set globals:
-- none

  function IS_TYPE_IN_ONE_USED_PACKAGE
          (TYPE_IN  : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
           return     BOOLEAN is

    USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR :=
           EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
    TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
           DDL_VARIABLES.FIRST_TYPE;
    COUNT : NATURAL := 0;
```

```
  begin
    while TYPE_DES /= null loop
      if (TYPE_DES.TYPE_KIND = A_TYPE or
          TYPE_DES.TYPE_KIND = A_SUBTYPE or
          TYPE_DES.TYPE_KIND = A_DERIVED) and
          TYPE_IN.NAME.all = TYPE_DES.FULL_NAME.NAME.all then
        USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
        while USED /= null loop
          if USED.NAME.all = TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all then
            COUNT := COUNT + 1;
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      TYPE_DES := TYPE_DES.NEXT_TYPE;
    end loop;
    if COUNT = 0 then
      return TRUE;
    else
      return FALSE;
    end if;
  end IS_TYPE_IN_ONE_USED_PACKAGE;

------------------------------------------------------------------------------
-- IS_TYPE_INT_ENUM_FLOAT_OR_STRING - if the type name is declared as an
--                                    integer, enumeration, float or string
--                                    return true else return false
-- set globals:
-- none

  function IS_TYPE_INT_ENUM_FLOAT_OR_STRING
          (TYPE_DES    : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
           return        BOOLEAN is
  begin
    return TYPE_DES.TYPE_IS.WHICH_TYPE /= REC_ORD;
  end IS_TYPE_INT_ENUM_FLOAT_OR_STRING;

------------------------------------------------------------------------------
-- IS_TABLE_IN_FROM_CLAUSE - if the table name is declared in the appropriate
--                           scope of the from clause return true else
--                           return false
-- set globals:
-- GLOBAL_FULL_NAME_DES - the full_name_descriptor of the table

  function IS_TABLE_IN_FROM_CLAUSE
          (TABLE : STRING)
           return  BOOLEAN is

    TABLE_DES       : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
```

```
    CORRELATION_DES : CORRELATION.NAME_DECLARED_ENTRY;

  begin
    GLOBAL_FULL_NAME_DES := null;
    FROM_CLAUSE.EXPOSES_NAME (TABLE, PARM_SCOPE, PARM_THIS_SCOPE_ONLY,
            TABLE_DES, CORRELATION_DES);
    if TABLE_DES = null then
      return FALSE;
    else
      GLOBAL_FULL_NAME_DES := TABLE_DES.FULL_NAME;
      return TRUE;
    end if;
  end IS_TABLE_IN_FROM_CLAUSE;

--------------------------------------------------------------------------
-- IS_COLUMN_IN_THIS_TABLE - if the column name is in the table name return
--                            true else return false
-- set globals:
-- GLOBAL_FULL_NAME_DES - access full name descriptor of column or null

  function IS_COLUMN_IN_THIS_TABLE
          (COLUMN  : STRING;
           TABLE   : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
           return    BOOLEAN is

    COMPONENT : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                TABLE.TYPE_IS.FIRST_COMPONENT;

  begin
    GLOBAL_FULL_NAME_DES := null;
    while COMPONENT /= null loop
      if COLUMN = STRING (COMPONENT.FULL_NAME.NAME.all) then
        GLOBAL_FULL_NAME_DES := COMPONENT.FULL_NAME;
        return TRUE;
      end if;
      COMPONENT := COMPONENT.NEXT_ONE;
    end loop;
    return FALSE;
  end IS_COLUMN_IN_THIS_TABLE;

--------------------------------------------------------------------------
-- IS_COLUMN_IN_THIS_CORRELATION_TABLE - if the column name is in the table
--                                        specified by the correlation name
--                                        return true else return false
-- set globals:
-- GLOBAL_FULL_NAME_DES - access_full_name_descriptor for column

  function IS_COLUMN_IN_THIS_CORRELATION_TABLE
          (COLUMN : STRING;
```

```
          CORR    : CORRELATION.NAME_DECLARED_ENTRY)
          return   BOOLEAN is

   TABLE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

 begin
   GLOBAL_FULL_NAME_DES := null;
   TABLE_DES := CORRELATION.TABLE_DECLARED_FOR (CORR);
   return IS_COLUMN_IN_THIS_TABLE (COLUMN, TABLE_DES.FULL_NAME);
 end IS_COLUMN_IN_THIS_CORRELATION_TABLE;

--------------------------------------------------------------------------
-- IS_COLUMN_IN_ONE_FROM_CLAUSE_TABLE - if the column name is in only one
--                                      table in the from caluse for the
--                                      appropriate scope return true else
--                                      return false
-- sets globals:
-- duplicated - false if column is not found more than once
--            - true if column is found more than once
-- GLOBAL_FULL_NAME_DES - null if not found
--            - if found it's the access full name descriptor of the column

 function IS_COLUMN_IN_ONE_FROM_CLAUSE_TABLE
         (COLUMN : STRING)
          return   BOOLEAN is
 begin
   GLOBAL_FULL_NAME_DES := null;
   DUPLICATED := FALSE;
   FROM_CLAUSE.MAKES_COLUMN_VISIBLE (COLUMN, PARM_SCOPE, PARM_THIS_SCOPE_ONLY,
           DUPLICATED, GLOBAL_FULL_NAME_DES);
   if DUPLICATED or
      GLOBAL_FULL_NAME_DES = null then
     return FALSE;
   else
     return TRUE;
   end if;
 end IS_COLUMN_IN_ONE_FROM_CLAUSE_TABLE;

--------------------------------------------------------------------------
-- IS_CORRELATION_IN_FROM_CLAUSE - if the correlation name is in the from
--                                 clause for the appropriate scope return
--                                 true else return false
-- set globals:
-- GLOBAL_CORRELATION_ENTRY

 function IS_CORRELATION_IN_FROM_CLAUSE
         (CORR  : STRING)
          return  BOOLEAN is
```

```
    TABLE_DES       : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    CORRELATION_DES : CORRELATION.NAME_DECLARED_ENTRY;

  begin
    GLOBAL_CORRELATION_ENTRY := null;
    FROM_CLAUSE.EXPOSES_NAME (CORR, PARM_SCOPE, PARM_THIS_SCOPE_ONLY,
            TABLE_DES, CORRELATION_DES);
    if CORRELATION_DES = null then
      return FALSE;
    else
      GLOBAL_CORRELATION_ENTRY := CORRELATION_DES;
      return TRUE;
    end if;
  end IS_CORRELATION_IN_FROM_CLAUSE;

---------------------------------------------------------------------------
-- IS_PACKAGE_WITHED - if the package name is withed by the current package
--                     return true else return false
-- set globals:
-- none

  function IS_PACKAGE_WITHED
          (PAK : STRING)
           return BOOLEAN is

    WITHED : DDL_DEFINITIONS.ACCESS_WITHED_UNIT_DESCRIPTOR :=
             EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_WITHED;

  begin
    while WITHED /= null loop
      if PAK = STRING (WITHED.SCHEMA_UNIT.NAME.all) then
        return TRUE;
      end if;
      WITHED := WITHED.NEXT_WITHED;
    end loop;
    return FALSE;
  end IS_PACKAGE_WITHED;

---------------------------------------------------------------------------
-- IS_ENUM_IN_THIS_PACKAGE - if the enumeration literal is declared by
--                           package PACK1.PACK2 return true else return
--                           false
-- set globals:
-- none

  function IS_ENUM_IN_THIS_PACKAGE
          (ENUM  : STRING;
           PAK1  : STRING;
           PAK2  : STRING)
```

```
            return  BOOLEAN is

    ENUM_DES : DDL_DEFINITIONS.ACCESS_ENUM_LIT_DESCRIPTOR :=
               DDL_VARIABLES.FIRST_ENUM_LIT;
    ENUM_FULL : DDL_DEFINITIONS.ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
    PAK_NAME  : STRING (1..250);
    PAK_LEN   : NATURAL;

  begin
    BUILD_FULL_PACKAGE (PAK1, PAK2, PAK_NAME, PAK_LEN);
    while ENUM_DES /= null loop
      if ENUM = STRING (ENUM_DES.NAME.all) then
        ENUM_FULL := ENUM_DES.FIRST_FULL_ENUM_LIT;
        while ENUM_FULL /= null loop
          if PAK_NAME (1..PAK_LEN) = STRING
                 (ENUM_FULL.TYPE_IS.FULL_NAME.FULL_PACKAGE_NAME.all) then
            return TRUE;
          end if;
          ENUM_FULL := ENUM_FULL.NEXT_LIT;
        end loop;
      end if;
      ENUM_DES := ENUM_DES.NEXT_ENUM_LIT;
    end loop;
    return FALSE;
  end IS_ENUM_IN_THIS_PACKAGE;

---------------------------------------------------------------------------
-- IS_VARIABLE_IN_ONE_USED_PACKAGES - if the variable is declared in exactly
--                                    one used package return true else return
--                                    false
-- set globals:
-- none

  function IS_VARIABLE_IN_ONE_USED_PACKAGE
          (VAR  : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
           return BOOLEAN is

    VAR_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
              DDL_VARIABLES.FIRST_VARIABLE;
    COUNT : NATURAL := 0;
    USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR;

  begin
    while VAR_DES /= null loop
      if VAR.NAME.all = VAR_DES.FULL_NAME.NAME.all then
        USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
        while USED /= null loop
          if VAR.FULL_PACKAGE_NAME.all = USED.NAME.all then
            COUNT := COUNT + 1;
```

```
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      VAR_DES := VAR_DES.NEXT_TYPE;
    end loop;
    if COUNT = 1 then
      return TRUE;
    else
      return FALSE;
    end if;
  end IS_VARIABLE_IN_ONE_USED_PACKAGE;

----------------------------------------------------------------------------
-- IS_VARIABLE_IN_THIS_PACKAGE - if the variable is declared by package
--                                 PACK return true else return false
-- set globals:
-- GLOBAL_FULL_NAME_DES - variable

  function IS_VARIABLE_IN_THIS_PACKAGE
          (VAR  : STRING;
           PAK  : STRING)
           return BOOLEAN is

    VAR_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
              DDL_VARIABLES.FIRST_VARIABLE;

  begin
    GLOBAL_FULL_NAME_DES := null;
    while VAR_DES /= null loop
      if VAR = STRING (VAR_DES.FULL_NAME.NAME.all) and
         PAK = STRING (VAR_DES.FULL_NAME.FULL_PACKAGE_NAME.all) then
         GLOBAL_FULL_NAME_DES := VAR_DES.FULL_NAME;
         return TRUE;
      end if;
      VAR_DES := VAR_DES.NEXT_TYPE;
    end loop;
    return FALSE;
  end IS_VARIABLE_IN_THIS_PACKAGE;

----------------------------------------------------------
-- FIND_ENUM_DES - given a string return the enum_lit_descriptor if there
--                 is one or else null
-- set globals:
-- none

    function FIND_ENUM_DES
            (ENUM_LIT : STRING)
             return     DDL_DEFINITIONS.ACCESS_ENUM_LIT_DESCRIPTOR is
```

```
    ENUM_DES : DDL_DEFINITIONS.ACCESS_ENUM_LIT_DESCRIPTOR :=
               DDL_VARIABLES.FIRST_ENUM_LIT;

  begin
    while ENUM_DES /= null loop
      if ENUM_LIT = STRING (ENUM_DES.NAME.all) then
        return ENUM_DES;
      end if;
      ENUM_DES := ENUM_DES.NEXT_ENUM_LIT;
    end loop;
    return null;
  end FIND_ENUM_DES;

----------------------------------------------------------------------
-- BUILD_ENUM_TYPE_LIST - start with the enumeration literal and find the
--                        enum_lit_descriptor, if not found return -1, then
--                        build a list of all of the full_enum_lit_descriptors
--                        that are visible considering the two possible
--                        package names and return the number of them
-- set globals:
-- GLOBAL_ENUM_TYPE_LIST - null if found none or the list if we
--                         found one or more

    function BUILD_ENUM_TYPE_LIST
            (ENUM_LIT   : STRING;
             PAK1       : STRING;
             PAK2       : STRING)
             return       INTEGER is

    ENUM_DES     : DDL_DEFINITIONS.ACCESS_ENUM_LIT_DESCRIPTOR;
    COUNT        : INTEGER;
    ENUM_FULL    : DDL_DEFINITIONS.ACCESS_FULL_ENUM_LIT_DESCRIPTOR;
    USED         : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR;
    FULL_PAK     : STRING (1..250);
    FULL_PAK_LEN : NATURAL;

    begin
      COUNT := -1;
      ENUM_DES := FIND_ENUM_DES (ENUM_LIT);
      if ENUM_DES = null then
        return COUNT;
      end if;
      COUNT := 0;
      BUILD_FULL_PACKAGE (PAK1, PAK2, FULL_PAK, FULL_PAK_LEN);
      GLOBAL_ENUM_TYPE_LIST := ENUMERATION.TYPE_LIST_CREATOR;
      ENUM_FULL := ENUM_DES.FIRST_FULL_ENUM_LIT;
      while ENUM_FULL /= null loop
        if FULL_PAK_LEN > 0 and then
           FULL_PAK (1..FULL_PAK_LEN) = STRING
```

```
                 (ENUM_FULL.TYPE_IS.FULL_NAME.FULL_PACKAGE_NAME.all) then
            ENUMERATION.TYPE_GOES_ON_LIST (ENUM_FULL.TYPE_IS.FULL_NAME,
                                  GLOBAL_ENUM_TYPE_LIST);
            COUNT := COUNT + 1;
          else
            USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
            while USED /= null loop
              if LIBRARY_UNIT_NAME_STRING(USED.NAME.all) =
                          ENUM_FULL.TYPE_IS.FULL_NAME.SCHEMA_UNIT.NAME.all then
                ENUMERATION.TYPE_GOES_ON_LIST (ENUM_FULL.TYPE_IS.FULL_NAME,
                                    GLOBAL_ENUM_TYPE_LIST);
                COUNT := COUNT + 1;
              end if;
              USED := USED.NEXT_USED;
            end loop;
          end if;
          ENUM_FULL := ENUM_FULL.NEXT_LIT;
        end loop;
        return COUNT;
      end BUILD_ENUM_TYPE_LIST;

  --------------------------------------------------------
  -- FIND_TYPE_DES - given a string that may be a type name return the
  --                 full_name_descriptor for a type if it's unambigous,
  --                 else return null.  Set duplicated if it's ambigous
  -- set globals:
  -- duplicated - false if found no more than one
  -- global_full_name_des - for type or null

      function FIND_TYPE_DES
              (TYPE_NAME : STRING)
               return      BOOLEAN is

      TYPE_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                 DDL_VARIABLES.FIRST_TYPE;
      USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR;

    begin
      DUPLICATED := FALSE;
      GLOBAL_FULL_NAME_DES := null;
      while TYPE_DES /= null loop
        if TYPE_NAME = STRING (TYPE_DES.FULL_NAME.NAME.all) then
          USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
          while USED /= null loop
            if USED.NAME.all = TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all then
              if GLOBAL_FULL_NAME_DES = null then
                GLOBAL_FULL_NAME_DES := TYPE_DES.FULL_NAME;
              else
                GLOBAL_FULL_NAME_DES := null;
```

```
              DUPLICATED := TRUE;
              return FALSE;
            end if;
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      TYPE_DES := TYPE_DES.NEXT_TYPE;
    end loop;
    if GLOBAL_FULL_NAME_DES /= null then
      return TRUE;
    else
      return FALSE;
    end if;
  end FIND_TYPE_DES;

--++++++++++  5
---------------------------------------------------------
-- FIND_VARIABLE_DES - given a string that may be a variable name return the
--                     full_name_descriptor for a variable if it's unambigous,
--                     else return null.  Set duplicated if it's ambigous
-- set globals:
-- duplicated - false if found no more than one
-- global_full_name_des - for variable or null
--                         variable if there is one or else null

    function FIND_VARIABLE_DES
           (VARIABLE : STRING)
            return     BOOLEAN is

    VAR_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
              DDL_VARIABLES.FIRST_VARIABLE;
    USED : DDL_DEFINITIONS.ACCESS_USED_PACKAGE_DESCRIPTOR;

  begin
    DUPLICATED := FALSE;
    GLOBAL_FULL_NAME_DES := null;
    while VAR_DES /= null loop
      if VARIABLE = STRING (VAR_DES.FULL_NAME.NAME.all) then
        USED := EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.FIRST_USED;
        while USED /= null loop
          if USED.NAME.all = VAR_DES.FULL_NAME.FULL_PACKAGE_NAME.all then
            if GLOBAL_FULL_NAME_DES = null then
              GLOBAL_FULL_NAME_DES := VAR_DES.FULL_NAME;
            else
              DUPLICATED := TRUE;
              GLOBAL_FULL_NAME_DES := null;
              return FALSE;
            end if;
```

```
          end if;
          USED := USED.NEXT_USED;
        end loop;
      end if;
      VAR_DES := VAR_DES.NEXT_TYPE;
    end loop;
    if GLOBAL_FULL_NAME_DES /= null then
      return TRUE;
    else
      return FALSE;
    end if;
  end FIND_VARIABLE_DES;

----------------------------------------------------------------------
-- COLUMN_NONE_NONE - see if we have a one part column name

  function COLUMN_NONE_NONE
          (COLUMN        : STRING)
           return          BOOLEAN is

  begin
    if IS_COLUMN_IN_ONE_FROM_CLAUSE_TABLE (COLUMN) then
      INFORMATION_OF_UNQUALIFIED_COLUMN.UNQUALIFIED_COLUMN :=
                                            GLOBAL_FULL_NAME_DES;
      if IS_PACKAGE_WITHED (COLUMN) then
        NAME_ERROR ("Column name may not be the same as any " &
                    "withed package name");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VALUE then
        NAME_ERROR ("Program value required - column name found");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
        NAME_ERROR ("Program variable required - column name found");
      else
        WHAT_KIND := OF_UNQUALIFIED_COLUMN;
        return TRUE;
      end if;
    else
      if DUPLICATED then
        NAME_ERROR ("Column name defined in more than one table");
      end if;
      return FALSE;
    end if;
  end COLUMN_NONE_NONE;

----------------------------------------------------------------------
-- ENUM_NONE_NONE - see if we have a one part enumeration literal

  function ENUM_NONE_NONE
          (ENUM_LIT : STRING)
           return     BOOLEAN is
```

```
    COUNT     : INTEGER;

  begin
    COUNT := BUILD_ENUM_TYPE_LIST (ENUM_LIT, "", "");
    INFORMATION_OF_ENUMERATION_LITERAL.ENUMERATION_TYPE_LIST :=
                                                         GLOBAL_ENUM_TYPE_LIST;
    if COUNT < 0 then
      return FALSE;
    elsif COUNT = 0 then
      return FALSE;
      --NAME_ERROR ("Enumeration literal is not visible from here");
    elsif ANY_CORRELATION_NAME (ENUM_LIT) then
      NAME_ERROR ("Enumeration literal may not be the same as " &
                  "any correlation name");
    elsif ANY_VARIABLE_FROM_USED_PACKAGES (ENUM_LIT) then
      NAME_ERROR ("Enumeration literal may not be the same as " &
                  "a variable name from any used package");
    elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
      NAME_ERROR ("Column specification required - enumeration literal found");
    elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
      NAME_ERROR ("Column name required - enumeration literal found");
    elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
      NAME_ERROR ("Program variable required - enumeration literal found");
    else
      WHAT_KIND := OF_ENUMERATION_LITERAL;
      return TRUE;
    end if;
  end ENUM_NONE_NONE;

-------------------------------------------------------------------------
-- TYPE_NONE_NONE - see if we have a one part type name

  function TYPE_NONE_NONE
          (TYPE_NAME : STRING)
           return      BOOLEAN is

  begin
    if FIND_TYPE_DES (TYPE_NAME) then
      INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE := GLOBAL_FULL_NAME_DES;
      if not IS_TYPE_IN_ONE_USED_PACKAGE
           (INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE) then
        NAME_ERROR ("Type name may be declared in only one used package");
      elsif IS_PACKAGE_WITHED (TYPE_NAME) then
        NAME_ERROR ("Type name may not be the same as " &
                    "any withed package name");
      elsif ANY_CORRELATION_NAME (TYPE_NAME) then
        NAME_ERROR ("Type name may not be the same as any correlation name");
      elsif not IS_TYPE_INT_ENUM_FLOAT_OR_STRING
           (INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE) then
```

```
          NAME_ERROR ("Type of integer, enumeration, floating point " &
                      "or string required");
        elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
          NAME_ERROR ("Column specification required - type found");
        elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
          NAME_ERROR ("Column name required - type found");
        elsif (PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
               not PARM_ALLOW_TYPE_CONVERSION) or
              (PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
               PARM_ALLOW_TYPE_CONVERSION and not TYPE_QUALIFICATION) then
          NAME_ERROR ("Program variable required - type found");
        else
          WHAT_KIND := OF_PROGRAM_TYPE;
          return TRUE;
        end if;
      else
        if DUPLICATED then
          NAME_ERROR ("Type is ambigous - qualification required");
        end if;
        return FALSE;
      end if;
    end TYPE_NONE_NONE;

  ----------------------------------------------------------------------
  -- VARIABLE_NONE_NONE - see if we have a one part variable name

    function VARIABLE_NONE_NONE
            (VAR   : STRING)
             return BOOLEAN is

    begin
      if FIND_VARIABLE_DES (VAR) then
        INFORMATION_OF_VARIABLE.VARIABLE_TYPE := GLOBAL_FULL_NAME_DES;
        if not IS_VARIABLE_IN_ONE_USED_PACKAGE
             (INFORMATION_OF_VARIABLE.VARIABLE_TYPE) then
          NAME_ERROR ("Variable names may be declared in only one used package");
        elsif IS_PACKAGE_WITHED (VAR) then
          NAME_ERROR ("Variable name may not be the same as " &
                      "any withed package name");
        elsif ANY_CORRELATION_NAME (VAR) then
          NAME_ERROR ("Variable name may not be the same as " &
                      "any correlation name");
        elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
          NAME_ERROR ("Column specification required - variable found");
        elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
          NAME_ERROR ("Column name required - variable found");
        else
          WHAT_KIND := OF_VARIABLE;
          return TRUE;
```

```
      end if;
    else
      if DUPLICATED then
        NAME_ERROR ("Variable name is ambigous - qualification required");
      end if;
      return FALSE;
    end if;
  end VARIABLE_NONE_NONE;

-----------------------------------------------------------------------
-- CORRELATION_COLUMN_NAME - see if we have a one part correlation name

  function CORRELATION_COLUMN_NONE
          (CORRELATION : STRING;
           COLUMN      : STRING)
           return        BOOLEAN is

  begin
    if IS_CORRELATION_IN_FROM_CLAUSE (CORRELATION) then
      INFORMATION_OF_CORRELATED_COLUMN.CORRELATION_NAME :=
                                      GLOBAL_CORRELATION_ENTRY;
      if IS_COLUMN_IN_THIS_CORRELATION_TABLE (COLUMN,
                INFORMATION_OF_CORRELATED_COLUMN.CORRELATION_NAME) then
        INFORMATION_OF_CORRELATED_COLUMN.CORRELATED_COLUMN :=
                                      GLOBAL_FULL_NAME_DES;
        if PARM_RESTRICT_SO = IS_COLUMN_NAME then
          NAME_ERROR ("Column name required - correlated column name found");
        elsif PARM_RESTRICT_SO = IS_PROGRAM_VALUE then
          NAME_ERROR ("Program value required - correlated column name found");
        elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
          NAME_ERROR ("Program variable required - " &
                      "correlated column name found");
        else
          WHAT_KIND := OF_CORRELATED_COLUMN;
          return TRUE;
        end if;
      else
        NAME_ERROR ("Column is not in table specified by correlation name");
      end if;
    else
      return FALSE;
    end if;
  end CORRELATION_COLUMN_NONE;
--++++++++++  6
-----------------------------------------------------------------------
-- PACKAGE_TYPE_NONE - see if we have a two part package-type name

  function PACKAGE_TYPE_NONE
          (PAK_NAME  : STRING;
```

```
          TYPE_NAME : STRING)
          return      BOOLEAN is

begin
  if IS_TYPE_IN_THIS_PACKAGE (TYPE_NAME, PAK_NAME, "") then
    INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE := GLOBAL_FULL_NAME_DES;
    if not IS_PACKAGE_WITHED (PAK_NAME) then
      return FALSE;
      --NAME_ERROR ("Qualified type is not visible - package is not withed");
    elsif ANY_CORRELATION_NAME (PAK_NAME) then
      NAME_ERROR ("Package name may not be the same as " &
                  "any correlation name");
    elsif not IS_TYPE_INT_ENUM_FLOAT_OR_STRING
                      (INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE) then
      NAME_ERROR ("Type of integer, enumeration, floating point " &
                  "or string required");
    elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
        NAME_ERROR ("Column specification required - qualified type found");
    elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
        NAME_ERROR ("Column name required - qualified type found");
    elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
              not PARM_ALLOW_TYPE_CONVERSION then
        NAME_ERROR ("Program variable required - qualified type found");
    elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
              PARM_ALLOW_TYPE_CONVERSION and TYPE_QUALIFICATION then
        NAME_ERROR ("Program variable required - qualified type found");
    else
      WHAT_KIND := OF_PROGRAM_TYPE;
      return TRUE;
    end if;
  else
    return FALSE;
  end if;
end PACKAGE_TYPE_NONE;

------------------------------------------------------------------------
-- PACKAGE_VARIABLE_NONE - see if we have a two part package_variable name

function PACKAGE_VARIABLE_NONE
         (PAK_NAME : STRING;
          VARIABLE : STRING)
          return     BOOLEAN is

begin
  if IS_VARIABLE_IN_THIS_PACKAGE (VARIABLE, PAK_NAME) then
    INFORMATION_OF_VARIABLE.VARIABLE_TYPE := GLOBAL_FULL_NAME_DES;
    if not IS_PACKAGE_WITHED (PAK_NAME) then
      return FALSE;
      --NAME_ERROR ("Variable is not visible - package not withed");
```

```
      elsif ANY_CORRELATION_NAME (PAK_NAME) then
        NAME_ERROR ("Package name may not be the same as " &
                    "any correlation name");
      elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
        NAME_ERROR ("Column specification required - " &
                    "qualified variable found");
      elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
        NAME_ERROR ("Column name required - qualified variable found");
      else
        WHAT_KIND := OF_VARIABLE;
        return TRUE;
      end if;
    else
      return FALSE;
    end if;
  end PACKAGE_VARIABLE_NONE;

-----------------------------------------------------------------------
-- TABLE_COLUMN_NONE - see if we have a two part table-column name

  function TABLE_COLUMN_NONE
          (TABLE    : STRING;
           COLUMN   : STRING)
           return     BOOLEAN is

  begin
    if IS_TABLE_IN_FROM_CLAUSE (TABLE) then
      INFORMATION_OF_QUALIFIED_COLUMN.TABLE := GLOBAL_FULL_NAME_DES;
      if IS_COLUMN_IN_THIS_TABLE (COLUMN,
                                  INFORMATION_OF_QUALIFIED_COLUMN.TABLE) then
        INFORMATION_OF_QUALIFIED_COLUMN.QUALIFIED_COLUMN :=
                                  GLOBAL_FULL_NAME_DES;
        if ANY_TYPE_FROM_USED_PACKAGES (TABLE) then
          NAME_ERROR ("Table name cannot be the same as " &
                      "any type name from any used package");
        elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
          NAME_ERROR ("Unqualified column name required - " &
                      "qualified column name found");
        elsif PARM_RESTRICT_SO = IS_PROGRAM_VALUE then
          NAME_ERROR ("Program value required - qualified column name found");
        elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
          NAME_ERROR ("Program variable required - " &
                      "qualified column name found");
        else
          WHAT_KIND := OF_QUALIFIED_COLUMN;
          return TRUE;
        end if;
      else
        NAME_ERROR ("Column is not in specified table");
```

```
    end if;
  else
    return FALSF;
  end if;
end TABLE_COLUMN_NONE;

------------------------------------------------------------------------
-- CONVERT_PACKAGE_TYPE - see if we have a three part convert_to-package_type

  function CONVERT_PACKAGE_TYPE
          (CONVERT_TO : STRING;
           PAK_NAME   : STRING;
           TYPE_NAME  : STRING)
           return       BOOLEAN is

    ADA_SQL_PAK : A_S;

  begin
    if CONVERT_TO = "CONVERT_TO" then
      if IS_TYPE_IN_THIS_PACKAGE (TYPE_NAME, PAK_NAME, "") then
        ADA_SQL_PAK := STRING_NULL;
      elsif IS_TYPE_IN_THIS_PACKAGE (TYPE_NAME, PAK_NAME, "ADA_SQL") then
        ADA_SQL_PAK := STRING_ADA_SQL;
      else
        NAME_ERROR ("Convert_to function must specify a type");
      end if;
      INFORMATION_OF_CONVERT_FUNCTION.CONVERT_TO_TYPE := GLOBAL_FULL_NAME_DES;
      if not IS_PACKAGE_WITHED (PAK_NAME) then
        NAME_ERROR ("Type not visible - package not withed");
      elsif IS_ENUM_IN_THIS_PACKAGE (TYPE_NAME, PAK_NAME, ADA_SQL_PAK.all) then
        NAME_ERROR ("Type name may not be the same as " &
                    "an enumeration literal in the same package");
      elsif ANY_CORRELATION_NAME (PAK_NAME) then
        NAME_ERROR ("Package name may not be the same as " &
                    "any correlation name");
      elsif not IS_TYPE_INT_ENUM_FLOAT_OR_STRING
                    (INFORMATION_OF_CONVERT_FUNCTION.CONVERT_TO_TYPE) then
        NAME_ERROR ("Type of integer, enumeration, floating point " &
                    "or string required");
      elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION and
              not PARM_ALLOW_TYPE_CONVERSION then
        NAME_ERROR ("Column specification required - " &
                    "convert to function found");
      elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
        NAME_ERROR ("Column name required - convert to function found");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VALUE then
        NAME_ERROR ("Program value required - convert to function found");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
        NAME_ERROR ("Program variable required - convert to function found");
```

```
      else
        WHAT_KIND := OF_CONVERT_FUNCTION;
        return TRUE;
      end if;
    else
      return FALSE;
    end if;
  end CONVERT_PACKAGE_TYPE;

---------------------------------------------------------------------------
-- PACKAGE_ADASQL_ENUM - see if we have a three part package-adasql-enumeration

  function PACKAGE_ADASQL_ENUM
          (PAK1 : STRING;
           PAK2 : STRING;
           ENUM : STRING)
           return BOOLEAN is

    COUNT     : INTEGER;

  begin
    COUNT := BUILD_ENUM_TYPE_LIST (ENUM, PAK1, PAK2);
    INFORMATION_OF_ENUMERATION_LITERAL.ENUMERATION_TYPE_LIST :=
                                                       GLOBAL_ENUM_TYPE_LIST;
    if COUNT < 0 then
      return FALSE;
    elsif COUNT = 0 then
      return FALSE;
      --NAME_ERROR ("Qualified enumeration literal not found");
    elsif not IS_PACKAGE_WITHED (PAK1) then
      return FALSE;
      --NAME_ERROR ("Enumeration literal not visible - package not withed");
    elsif ANY_CORRELATION_NAME (PAK1) then
      NAME_ERROR ("Package name may not be the same as any correlation name");
    elsif ANY_TYPE_FROM_THIS_PACKAGE (ENUM, PAK1, PAK2) then
      NAME_ERROR ("Enumeration literal may not be the same as " &
                  "any type name from the same package");
    elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
      NAME_ERROR ("Column specification required - " &
                  "qualified enumeration literal found");
    elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
      NAME_ERROR ("Column name required - " &
                  "qualified enumeration literal found");
    elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE then
      NAME_ERROR ("Program variable required - " &
                  "qualified enumeration literal found");
    else
      WHAT_KIND := OF_ENUMERATION_LITERAL;
      return TRUE;
```

```
    end if;
  end PACKAGE_ADASQL_ENUM;

------------------------------------------------------------------------
-- PACKAGE_ADASQL_TYPE - see if we have a three part package-adasql-type

  function PACKAGE_ADASQL_TYPE
          (PAK1      : STRING;
           PAK2      : STRING;
           TYPE_NAME : STRING)
           return      BOOLEAN is

  begin
    if IS_TYPE_IN_THIS_PACKAGE (TYPE_NAME, PAK1, PAK2) then
      INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE := GLOBAL_FULL_NAME_DES;
      if not IS_PACKAGE_WITHED (PAK1) then
        return FALSE;
        --NAME_ERROR ("Type not visible - package not withed");
      elsif ANY_CORRELATION_NAME (PAK1) then
        NAME_ERROR ("Package name may not be the same as " &
                    "any correlation name");
      elsif ANY_ENUM_LIT_FROM_THIS_PACKAGE (TYPE_NAME, PAK1, PAK2) then
        NAME_ERROR ("Type name may not be the same as " &
                    "an enumeration literal in the same package");
      elsif not IS_TYPE_INT_ENUM_FLOAT_OR_STRING
                    (INFORMATION_OF_PROGRAM_TYPE.PROGRAM_TYPE) then
        NAME_ERROR ("Type of integer, enumeration, floating point " &
                    "or string required");
      elsif PARM_RESTRICT_SO = IS_COLUMN_SPECIFICATION then
        NAME_ERROR ("Column specification required - qualified type found");
      elsif PARM_RESTRICT_SO = IS_COLUMN_NAME then
        NAME_ERROR ("Column name required - qualified type found");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
            not PARM_ALLOW_TYPE_CONVERSION then
        NAME_ERROR ("Program variable required - qualified type found");
      elsif PARM_RESTRICT_SO = IS_PROGRAM_VARIABLE and
            PARM_ALLOW_TYPE_CONVERSION and TYPE_QUALIFICATION then
        NAME_ERROR ("Program variable required - type qualification found");
      else
        WHAT_KIND := OF_PROGRAM_TYPE;
        return TRUE;
      end if;
    else
      return FALSE;
    end if;
  end PACKAGE_ADASQL_TYPE;
--++++++++++  7
-----------------------------------------------------------------------
  function AT_CURRENT_INPUT_POINT
```

```
        (SCOPE                  : FROM_CLAUSE.INFORMATION;
         RESTRICT_SO            : KIND_RESTRICTION;
         THIS_SCOPE_ONLY        : BOOLEAN;
         ALLOW_TYPE_CONVERSION  : BOOLEAN;
         REPORT_ERRORS          : BOOLEAN := TRUE)
         return                   INFORMATION is

  PARTS_COUNT : NATURAL := 0;
  PART_1      : STRING (1..250) := (others => ' ');
  PART_1_LEN  : NATURAL := 0;
  PART_2      : STRING (1..250) := (others => ' ');
  PART_2_LEN  : NATURAL := 0;
  PART_3      : STRING (1..250) := (others => ' ');
  PART_3_LEN  : NATURAL := 0;
  TOKEN_COUNT : POSITIVE := 1;

begin
  PARM_SCOPE                    := SCOPE;
  PARM_RESTRICT_SO              := RESTRICT_SO;
  PARM_THIS_SCOPE_ONLY          := THIS_SCOPE_ONLY;
  PARM_ALLOW_TYPE_CONVERSION    := ALLOW_TYPE_CONVERSION;
  PARM_REPORT_ERRORS            := REPORT_ERRORS;
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  GET_NAME_PARTS (PARTS_COUNT, PART_1, PART_1_LEN, PART_2, PART_2_LEN,
                  PART_3, PART_3_LEN, TYPE_QUALIFICATION, TOKEN_COUNT);
  if PARTS_COUNT = 1 then
    if PARM_RESTRICT_SO = IS_PROGRAM_VALUE then
      if ENUM_NONE_NONE (PART_1 (1..PART_1_LEN)) then
        INFORMATION_OF_ENUMERATION_LITERAL.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_ENUMERATION_LITERAL;
      end if;
    end if;
    if COLUMN_NONE_NONE (PART_1 (1..PART_1_LEN)) then
      INFORMATION_OF_UNQUALIFIED_COLUMN.NUMBER_OF_TOKENS := TOKEN_COUNT;
      return INFORMATION_OF_UNQUALIFIED_COLUMN;
    elsif ENUM_NONE_NONE (PART_1 (1..PART_1_LEN)) then
      INFORMATION_OF_ENUMERATION_LITERAL.NUMBER_OF_TOKENS := TOKEN_COUNT;
      return INFORMATION_OF_ENUMERATION_LITERAL;
    elsif TYPE_NONE_NONE (PART_1 (1..PART_1_LEN)) then
      INFORMATION_OF_PROGRAM_TYPE.NUMBER_OF_TOKENS := TOKEN_COUNT;
      return INFORMATION_OF_PROGRAM_TYPE;
    elsif VARIABLE_NONE_NONE (PART_1 (1..PART_1_LEN)) then
      INFORMATION_OF_VARIABLE.NUMBER_OF_TOKENS := TOKEN_COUNT;
      return INFORMATION_OF_VARIABLE;
    end if;
  elsif PARTS_COUNT = 2 then
    if CORRELATION_COLUMN_NONE (PART_1 (1..PART_1_LEN),
                                PART_2 (1..PART_2_LEN)) then
      INFORMATION_OF_CORRELATED_COLUMN.NUMBER_OF_TOKENS := TOKEN_COUNT;
```

```
        return INFORMATION_OF_CORRELATED_COLUMN;
      elsif PACKAGE_TYPE_NONE (PART_1 (1..PART_1_LEN),
                               PART_2 (1..PART_2_LEN)) then
        INFORMATION_OF_PROGRAM_TYPE.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_PROGRAM_TYPE;
      elsif PACKAGE_VARIABLE_NONE (PART_1 (1..PART_1_LEN),
                                   PART_2 (1..PART_2_LEN)) then
        INFORMATION_OF_VARIABLE.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_VARIABLE;
      elsif TABLE_COLUMN_NONE (PART_1 (1..PART_1_LEN),
                               PART_2 (1..PART_2_LEN)) then
        INFORMATION_OF_QUALIFIED_COLUMN.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_QUALIFIED_COLUMN;
      end if;
    elsif PARTS_COUNT = 3 then
      if CONVERT_PACKAGE_TYPE (PART_1 (1..PART_1_LEN),
                               PART_2 (1..PART_2_LEN),
                               PART_3 (1..PART_3_LEN)) then
        INFORMATION_OF_CONVERT_FUNCTION.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_CONVERT_FUNCTION;
      elsif PACKAGE_ADASQL_ENUM (PART_1 (1..PART_1_LEN),
                                 PART_2 (1..PART_2_LEN),
                                 PART_3 (1..PART_3_LEN)) then
        INFORMATION_OF_ENUMERATION_LITERAL.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_ENUMERATION_LITERAL;
      elsif PACKAGE_ADASQL_TYPE (PART_1 (1..PART_1_LEN),
                                 PART_2 (1..PART_2_LEN),
                                 PART_3 (1..PART_3_LEN)) then
        INFORMATION_OF_PROGRAM_TYPE.NUMBER_OF_TOKENS := TOKEN_COUNT;
        return INFORMATION_OF_PROGRAM_TYPE;
      end if;
    end if;
    NAME_ERROR ("Identifier has no valid meaning in this context");
  end AT_CURRENT_INPUT_POINT;
end NAME;
```

3.11.64 package semans.ada

```
-- semans.ada - miscellaneous routines for semantic processing

with ADA_SQL_FUNCTION_DEFINITIONS, DDL_DEFINITIONS, GENERATED_FUNCTIONS,
 LEXICAL_ANALYZER, RESULT, SELEC;
package SEMANTICALLY is

  type LOCATION_RESTRICTION is ( ADA_VALUE , PROGRAM_VALUE , ANY_VALUE );

  type SQL_OPERATIONS is array ( LEXICAL_ANALYZER.DELIMITER_KIND ) of
   ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;

  BINARY_SQL_OPERATION : constant SQL_OPERATIONS :=
```

```
  ( ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- AMPERSAND
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- APOSTROPHE
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- LEFT_PARENTHESIS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- RIGHT_PARENTHESIS
    ADA_SQL_FUNCTION_DEFINITIONS.O_TIMES,       -- STAR
    ADA_SQL_FUNCTION_DEFINITIONS.O_PLUS,        -- PLUS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- COMMA
    ADA_SQL_FUNCTION_DEFINITIONS.O_MINUS,       -- HYPHEN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- DOT
    ADA_SQL_FUNCTION_DEFINITIONS.O_DIVIDE,      -- SLASH
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- COLON
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- SEMICOLON
    ADA_SQL_FUNCTION_DEFINITIONS.O_LT,          -- LESS_THAN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- EQUAL
    ADA_SQL_FUNCTION_DEFINITIONS.O_GT,          -- GREATER_THAN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- VERTICAL_BAR
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- ARROW
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- DOUBLE_DOT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- DOUBLE_STAR
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- ASSIGNMENT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- INEQUALITY
    ADA_SQL_FUNCTION_DEFINITIONS.O_GE,          -- GREATER_THAN_OR_EQUAL
    ADA_SQL_FUNCTION_DEFINITIONS.O_LE,          -- LESS_THAN_OR_EQUAL
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- LEFT_LABEL_BRACKET
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,     -- RIGHT_LABEL_BRACKET
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP );   -- BOX

-- SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS combines the types of two
-- operands (LEFT and RIGHT) and returns the combined type ("most known and
-- database-ish") in RETURN_TYPE.  If the types are not comparable, a semantic
-- error is printed for the given TOKEN, and the right type is returned as the
-- RETURN_TYPE (which is hopefully good enough to continue processing with).
-- The status of the comparability check is returned as COMPARABLE.

  procedure VALIDATE_COMPARABLE_OPERANDS
            ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              LEFT,
              RIGHT       : in  RESULT.DESCRIPTOR;
              RETURN_TYPE : out RESULT.DESCRIPTOR;
              COMPARABLE  : out RESULT.COMPARABILITY );

  procedure VALIDATE_COMPARABLE_OPERANDS
            ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              LEFT        : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
              RIGHT       : in  RESULT.DESCRIPTOR;
              RETURN_TYPE : out RESULT.DESCRIPTOR;
              COMPARABLE  : out RESULT.COMPARABILITY );

-- SEMANTICALLY.VALIDATE_DATABASE_VALUE_USED prints an error message for the
```

```
-- given TOKEN if the SAW_DATABASE_VALUE flag is not TRUE.

  procedure VALIDATE_DATABASE_VALUE_USED
            ( TOKEN              : LEXICAL_ANALYZER.LEXICAL_TOKEN;
              SAW_DATABASE_VALUE : BOOLEAN );

-- SEMANTICALLY.STRONGLY_TYPE returns the ACCESS_TYPE_DESCRIPTOR corresponding
-- to the (possibly unknown) given RETURN_TYPE.  The appropriate STANDARD
-- types are used for unknown types; null is returned for unknown enumeration
-- types, and is caught at the point of error, so that it will not propagate
-- upwards through routine returns and cause additional errors.

  function STRONGLY_TYPE ( RETURN_TYPE : RESULT.DESCRIPTOR )
   return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

-- SEMANTICALLY.VALIDATE_STRONGLY_TYPED behaves the same as SEMANTICALLY.-
-- STRONGLY_TYPE, except that it reports a semantic error message for an
-- unknown enumeration type.

  function VALIDATE_STRONGLY_TYPED
           ( TOKEN              : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
             RETURN_TYPE        : in  RESULT.DESCRIPTOR )
   return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

-- SEMANTICALLY.MAKE_BINARY_OPERATION generates the appropriate binary
-- OPERATION, returning the given RESULT_KIND.  The operands are strongly
-- typed with type STRONG_TYPE, and are either program or database values
-- according to LEFT_PARAMETER and RIGHT_PARAMETER.

  procedure MAKE_BINARY_OPERATION
            ( OPERATION       : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
              STRONG_TYPE     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
              LEFT_PARAMETER,
              RIGHT_PARAMETER : RESULT.DESCRIPTOR;
              RESULT_KIND     : GENERATED_FUNCTIONS.OPERAND_KIND );

-- SEMANTICALLY.GET_SELECT_WORD sets SELECT_SEEN to TRUE if the given token is
-- a select word, otherwise sets SELECT_SEEN to FALSE.  If SELECT_SEEN, then
-- SELECT_TYPE indicates the particular select word used.

 procedure GET_SELECT_WORD
           ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
             SELECT_SEEN : out BOOLEAN;
             SELECT_TYPE : out SELEC.ROUTINE_NAME );

end SEMANTICALLY;
```

3.11.65 package semanb.ada

```
-- semanb.ada - miscellaneous routines for semantic processing
```

```
with DDL_DEFINITIONS, GENERATED_FUNCTIONS, LEXICAL_ANALYZER, PREDEFINED_TYPE,
 RESULT;
 use DDL_DEFINITIONS, GENERATED_FUNCTIONS, LEXICAL_ANALYZER, RESULT;
package body SEMANTICALLY is

  procedure VALIDATE_COMPARABLE_OPERANDS
            ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              LEFT,
              RIGHT       : in  RESULT.DESCRIPTOR;
              RETURN_TYPE : out RESULT.DESCRIPTOR;
              COMPARABLE  : out RESULT.COMPARABILITY ) is
    OUR_COMPARABLE : RESULT.COMPARABILITY;
  begin
    RESULT.COMBINED_TYPE ( LEFT , RIGHT , RETURN_TYPE , OUR_COMPARABLE );
    COMPARABLE := OUR_COMPARABLE;
    if OUR_COMPARABLE = RESULT.IS_NOT_COMPARABLE then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( TOKEN , "Operands not comparable" );
      RETURN_TYPE := RIGHT;
    end if;
  end VALIDATE_COMPARABLE_OPERANDS;

  procedure VALIDATE_COMPARABLE_OPERANDS
            ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              LEFT        : in  DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
              RIGHT       : in  RESULT.DESCRIPTOR;
              RETURN_TYPE : out RESULT.DESCRIPTOR;
              COMPARABLE  : out RESULT.COMPARABILITY ) is
    OUR_COMPARABLE : RESULT.COMPARABILITY;
  begin
    RESULT.COMBINED_TYPE ( LEFT , RIGHT , RETURN_TYPE , OUR_COMPARABLE );
    COMPARABLE := OUR_COMPARABLE;
    if OUR_COMPARABLE = RESULT.IS_NOT_COMPARABLE then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( TOKEN , "Operands not comparable" );
      RETURN_TYPE := RIGHT;
    end if;
  end VALIDATE_COMPARABLE_OPERANDS;

  procedure VALIDATE_DATABASE_VALUE_USED
            ( TOKEN              : LEXICAL_ANALYZER.LEXICAL_TOKEN;
              SAW_DATABASE_VALUE : BOOLEAN ) is
  begin
    if not SAW_DATABASE_VALUE then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( TOKEN , "Operand from database required" );
    end if;
  end VALIDATE_DATABASE_VALUE_USED;
```

```
function STRONGLY_TYPE ( RETURN_TYPE : RESULT.DESCRIPTOR )
 return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
begin
  if RETURN_TYPE.TYPE_IS = RESULT.IS_KNOWN then
    return RETURN_TYPE.KNOWN_TYPE;
  else
    case RETURN_TYPE.UNKNOWN_TYPE.CLASS is
      when DDL_DEFINITIONS.INT_EGER =>
        return PREDEFINED_TYPE.STANDARD.INTEGER;
      when DDL_DEFINITIONS.FL_OAT =>
        return PREDEFINED_TYPE.STANDARD.FLOAT;
      when DDL_DEFINITIONS.STR_ING =>
        return PREDEFINED_TYPE.STANDARD.STRING;
      when others =>
        return null;
    end case;
  end if;
end STRONGLY_TYPE;

function VALIDATE_STRONGLY_TYPED
         ( TOKEN              : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
           RETURN_TYPE        : in  RESULT.DESCRIPTOR )
 return DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR is
  T : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
   STRONGLY_TYPE ( RETURN_TYPE );
begin
  if T = null then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN , "Type of enumeration operand(s) is ambiguous" );
  end if;
  return T;
end VALIDATE_STRONGLY_TYPED;

function PARAMETER_KIND ( PARAMETER : RESULT.DESCRIPTOR )
 return GENERATED_FUNCTIONS.OPERAND_KIND is
begin
  if PARAMETER.LOCATION = RESULT.IN_DATABASE then
    return GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT;
  else
    return GENERATED_FUNCTIONS.O_USER_TYPE;
  end if;
end PARAMETER_KIND;

procedure MAKE_BINARY_OPERATION
          ( OPERATION       : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
            STRONG_TYPE     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            LEFT_PARAMETER,
            RIGHT_PARAMETER : RESULT.DESCRIPTOR;
            RESULT_KIND     : GENERATED_FUNCTIONS.OPERAND_KIND ) is
```

```
    LEFT_PARAMETER_KIND   : GENERATED_FUNCTIONS.OPERAND_KIND;
    RIGHT_PARAMETER_KIND  : GENERATED_FUNCTIONS.OPERAND_KIND;
    RESULT_TYPE           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
  begin
    LEFT_PARAMETER_KIND := PARAMETER_KIND ( LEFT_PARAMETER );
    RIGHT_PARAMETER_KIND := PARAMETER_KIND ( RIGHT_PARAMETER );
    if RESULT_KIND = GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT then
      RESULT_TYPE := STRONG_TYPE.FULL_NAME;
    else
      RESULT_TYPE := null;
    end if;
    GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
    ( OPERATION , LEFT_PARAMETER_KIND , STRONG_TYPE.FULL_NAME ,
      RIGHT_PARAMETER_KIND , STRONG_TYPE.FULL_NAME , RESULT_KIND ,
      RESULT_TYPE );
  end MAKE_BINARY_OPERATION;

  procedure GET_SELECT_WORD
           ( TOKEN       : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
             SELECT_SEEN : out BOOLEAN;
             SELECT_TYPE : out SELEC.ROUTINE_NAME ) is
  begin
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
      begin
        SELECT_TYPE := SELEC.ROUTINE_NAME'VALUE ( TOKEN.ID.all );
        SELECT_SEEN := TRUE;
      exception
        when CONSTRAINT_ERROR => SELECT_SEEN := FALSE;
      end;
    else
      SELECT_SEEN := FALSE;
    end if;
  end GET_SELECT_WORD;

end SEMANTICALLY;
```

3.11.66 package posts.ada

```
-- posts.ada - produce generated package (specification and body).

package POST_PROCESS is

   procedure GENERATE_PACKAGE
      (GENERATED_PACKAGE_FILENAME : in STRING);

end POST_PROCESS;
```

3.11.67 package postb.ada

```
-- postb.ada - produce generated package (specification and body).

with TEXT_PRINT, TEXT_IO, EXTRA_DEFINITIONS, WITH_REQUIRED, UNQUALIFIED_NAME,
     INDEX_SUBTYPE, DATABASE_TYPE, QUALIFIED_NAME, CORRELATION, CONVERT_TO,
     CONVERT_COMPONENT_TO_CHARACTER, PROGRAM_CONVERSION,
     CONVERT_CHARACTER_TO_COMPONENT, GENERATED_FUNCTIONS, INDICATOR,
     PREDEFINED, INTO, SELEC, LEXICAL_ANALYZER;
use  TEXT_PRINT;
package body POST_PROCESS is

   OUTPUT_FILE : TEXT_IO.FILE_TYPE;

procedure CREATE_FILE
   (FILENAME : in STRING) is
begin
   TEXT_IO.CREATE (OUTPUT_FILE, TEXT_IO.OUT_FILE, FILENAME);
exception
   when others =>
      LEXICAL_ANALYZER.REPORT_FATAL_ERROR
         ("Unable to create generated package file: " & FILENAME);
end CREATE_FILE;

procedure CLOSE_FILE is
begin
   TEXT_IO.CLOSE (OUTPUT_FILE);
   TEXT_IO.SET_OUTPUT (TEXT_IO.STANDARD_OUTPUT);
exception
   when others =>
      LEXICAL_ANALYZER.REPORT_FATAL_ERROR
         ("Unable to close generated package file");
end CLOSE_FILE;

procedure INITIALIZE_TEXT_PRINT is
   LINE : TEXT_PRINT.LINE_TYPE;
begin
   TEXT_IO.SET_OUTPUT (OUTPUT_FILE);
   TEXT_PRINT.CREATE_LINE (LINE, 80);
   TEXT_PRINT.SET_LINE (LINE);
   TEXT_PRINT.SET_CONTINUATION_INDENT (2);
   TEXT_PRINT.SET_INDENT (0);
end INITIALIZE_TEXT_PRINT;

procedure GENERATE_PACKAGE_SPECIFICATION is
begin
   WITH_REQUIRED.POST_PROCESSING;
   SET_INDENT (0);
```

```
PRINT ("pragma ELABORATE (ADA_SQL_FUNCTIONS);");
PRINT_LINE;
PRINT ("package ");
PRINT (STRING(EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.NAME.all) &
       "_ADA_SQL");
PRINT (" is");
PRINT_LINE;
BLANK_LINE;
SET_INDENT (2);
PRINT ("use ADA_SQL_FUNCTIONS.CONVERT;");
PRINT_LINE;
BLANK_LINE;
PRINT ("NO_UPDATE_ERROR : exception renames");
PRINT (" ADA_SQL_FUNCTIONS.NO_UPDATE_ERROR;");
PRINT_LINE;
PRINT ("NOT_FOUND_ERROR : exception renames");
PRINT (" ADA_SQL_FUNCTIONS.NOT_FOUND_ERROR;");
PRINT_LINE;
PRINT ("INTERNAL_ERROR : exception renames");
PRINT (" ADA_SQL_FUNCTIONS.INTERNAL_ERROR;");
PRINT_LINE;
PRINT ("UNIQUE_ERROR : exception renames");
PRINT (" ADA_SQL_FUNCTIONS.UNIQUE_ERROR;");
PRINT_LINE;
BLANK_LINE;
PRINT ("procedure OPEN_DATABASE"); PRINT_LINE;
PRINT ("          (DATABASE_NAME : in STANDARD.STRING;"); PRINT_LINE;
PRINT ("           PASSWORD      : in STANDARD.STRING)"); PRINT_LINE;
PRINT (" renames ADA_SQL_FUNCTIONS.OPEN_DATABASE;"); PRINT_LINE;
BLANK_LINE;
PRINT ("procedure EXIT_DATABASE renames ADA_SQL_FUNCTIONS.EXIT_DATABASE;");
PRINT_LINE;
BLANK_LINE;
PRINT ("package ADA_SQL is");
PRINT_LINE;
BLANK_LINE;
UNQUALIFIED_NAME.POST_PROCESSING_1;
INDEX_SUBTYPE.POST_PROCESSING;
DATABASE_TYPE.POST_PROCESSING_TO_PRODUCE_TYPE_DECLARATIONS;
DATABASE_TYPE.POST_PROCESSING_TO_PRODUCE_UNQUALIFIED_USE_CLAUSE;
QUALIFIED_NAME.POST_PROCESSING_1;
SET_INDENT (2);
PRINT ("end ADA_SQL;");
PRINT_LINE;
BLANK_LINE;
DATABASE_TYPE.POST_PROCESSING_TO_PRODUCE_QUALIFIED_USE_CLAUSE;
QUALIFIED_NAME.POST_PROCESSING_2;
UNQUALIFIED_NAME.POST_PROCESSING_2;
CORRELATION.NAME_POST_PROCESS;
```

```
   CONVERT_TO.POST_PROCESSING;
   CONVERT_COMPONENT_TO_CHARACTER.SPEC_POST_PROCESSING;
   PROGRAM_CONVERSION.POST_PROCESSING;
   CONVERT_CHARACTER_TO_COMPONENT.SPEC_POST_PROCESSING;
   GENERATED_FUNCTIONS.POST_PROCESSING;
   INDICATOR.POST_PROCESSING;
   PREDEFINED.TEXT_POST_PROCESSING_1;
   INTO.POST_PROCESSING;
   SELEC.POST_PROCESSING_1;
   SET_INDENT (2);
   PRINT ("function KLUDGE_FOR_VAX_ADA_BUG ");
   PRINT_LINE;
   PRINT ("   ( L : ADA_SQL_FUNCTIONS.SQL_OBJECT )");
   PRINT_LINE;
   PRINT ("   return ADA_SQL_FUNCTIONS.SQL_OBJECT renames CONVERT_R;");
   PRINT_LINE;
   BLANK_LINE;
   SET_INDENT (0);
   PRINT ("end ");
   PRINT (STRING(EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.NAME.all) &
          "_ADA_SQL;");
   PRINT_LINE;
   BLANK_LINE;
end GENERATE_PACKAGE_SPECIFICATION;

procedure GENERATE_PACKAGE_BODY is
begin
   SET_INDENT (0);
   PRINT ("package body ");
   PRINT (STRING(EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.NAME.all) &
          "_ADA_SQL");
   PRINT (" is");
   PRINT_LINE;
   BLANK_LINE;
   CORRELATION.NAME_POST_PROCESS_KLUDGE;
   CONVERT_COMPONENT_TO_CHARACTER.BODY_POST_PROCESSING;
   CONVERT_CHARACTER_TO_COMPONENT.BODY_POST_PROCESSING;
   PREDEFINED.TEXT_POST_PROCESSING_2;
   SELEC.POST_PROCESSING_2;
   SET_INDENT (0);
   PRINT ("end ");
   PRINT (STRING(EXTRA_DEFINITIONS.CURRENT_SCHEMA_UNIT.NAME.all) &
          "_ADA_SQL;");
   PRINT_LINE;
end GENERATE_PACKAGE_BODY;

procedure GENERATE_PACKAGE
   (GENERATED_PACKAGE_FILENAME : in STRING) is
```

```
begin
   CREATE_FILE (GENERATED_PACKAGE_FILENAME);
   INITIALIZE_TEXT_PRINT;
   GENERATE_PACKAGE_SPECIFICATION;
   GENERATE_PACKAGE_BODY;
   CLOSE_FILE;
end GENERATE_PACKAGE;

end POST_PROCESS;
```

3.11.68 package syntacs.ada

```
-- syntacs.ada - miscellaneous syntactic processing routines

with LEXICAL_ANALYZER, NAME;
package SYNTACTICALLY is

-- SYNTACTICALLY.GOBBLE_NAME eats the tokens comprising a name (as defined in
-- names.ada) given by its NAME.INFORMATION.

  procedure GOBBLE_NAME ( N : NAME.INFORMATION );

-- SYNTACTICALLY.IS_INTEGER returns TRUE or FALSE depending on whether the
-- given LEXICAL_TOKEN, which represents a NUMERIC_LITERAL, represents an
-- integer (no decimal point).

  function IS_INTEGER ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN )
   return BOOLEAN;

-- SYNTACTICALLY.IS_DELIMITER returns TRUE iff the first look-ahead token is
-- the delimiter provided.
  function IS_DELIMITER (DELIM : LEXICAL_ANALYZER.DELIMITER_KIND )
   return BOOLEAN;

-- SYNTACTICALLY.IS_IDENTIFIER returns TRUE iff the first look-ahead token is
-- an identifier.
  function IS_IDENTIFIER return BOOLEAN;

-- SYNTACTICALLY.IS_RESERVED_WORD returns TRUE iff the first look-ahead token is
-- the reserved word provided.
  function IS_RESERVED_WORD (WORD : LEXICAL_ANALYZER.RESERVED_WORD_KIND)
   return BOOLEAN;

-- SYNTACTICALLY.PROCESS_DELIMITER makes sure that the given delimiter is the
-- NEXT_TOKEN, reporting a syntax error if that is not so.  (It gobbles the
-- token if OK.)

  procedure PROCESS_DELIMITER ( DELIM : LEXICAL_ANALYZER.DELIMITER_KIND );

-- SYNTACTICALLY.PROCESS_RESERVED_WORD makes sure that the given reserved word
```

```
-- is the NEXT_TOKEN, reporting a syntax error if that is not so.  (It gobbles
-- the token if OK.)

  procedure PROCESS_RESERVED_WORD
            ( WORD : LEXICAL_ANALYZER.RESERVED_WORD_KIND );

-- SYNTACTICALLY.PROCESS_KEYWORD makes sure that the given identifier is the
-- NEXT_TOKEN, reporting a syntax error if that is not so.  (It gobbles the
-- token if OK.)

  procedure PROCESS_KEYWORD ( WORD : STRING );

-- SKIP_SELECT_CLAUSE naively skips over (1) a select word, (2) an opening
-- parenthesis, and (3) a select list.  On call, the next token is (1), which
-- is known to be valid, since we dispatched here.  (2) is validated, and the
-- RESTORE_SKIPPED_TOKENS pointer is left after (2).  (3) is skipped by merely
-- searching for a semicolon or FROM -- a semicolon is an error, FROM ends the
-- skip.

  procedure SKIP_SELECT_CLAUSE;

end SYNTACTICALLY;
```

3.11.69 package syntacb.ada

```
-- syntacb.ada - miscellaneous syntactic processing routines

with LEXICAL_ANALYZER, NAME;
 use LEXICAL_ANALYZER;
package body SYNTACTICALLY is

  procedure GOBBLE_NAME ( N : NAME.INFORMATION ) is
  begin
    for I in 1 .. N.NUMBER_OF_TOKENS loop
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    end loop;
  end GOBBLE_NAME;

  function IS_INTEGER ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN )
   return BOOLEAN is
  begin
    for I in TOKEN.IMAGE'RANGE loop
      if TOKEN.IMAGE(I) = '.' then
        return FALSE;
      end if;
    end loop;
    return TRUE;
  end IS_INTEGER;

  function IS_DELIMITER (DELIM : LEXICAL_ANALYZER.DELIMITER_KIND )
```

```
 return BOOLEAN is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
           LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
     TOKEN.DELIMITER = DELIM then
    return TRUE;
  end if;
  return FALSE;
end IS_DELIMITER;

function IS_IDENTIFIER return BOOLEAN is
begin
  if LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN.KIND =
     LEXICAL_ANALYZER.IDENTIFIER then
     return TRUE;
  end if;
  return FALSE;
end IS_IDENTIFIER;

function IS_RESERVED_WORD (WORD : LEXICAL_ANALYZER.RESERVED_WORD_KIND)
 return BOOLEAN is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
           LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.RESERVED_WORD and then
     TOKEN.RESERVED_WORD = WORD then
     return TRUE;
  end if;
  return FALSE;
end IS_RESERVED_WORD;

procedure PROCESS_KEYWORD ( WORD : STRING ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  if TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER or else
   TOKEN.ID.all /= WORD then
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
    ( TOKEN , "Expecting " & WORD );
  end if;
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
end PROCESS_KEYWORD;


procedure PROCESS_DELIMITER ( DELIM : LEXICAL_ANALYZER.DELIMITER_KIND ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
```

```
begin
  if TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
   TOKEN.DELIMITER /= DELIM then
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
    ( TOKEN , "Expecting " & LEXICAL_ANALYZER.DELIMITER_KIND'IMAGE(DELIM) );
  end if;
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
end PROCESS_DELIMITER;

procedure PROCESS_RESERVED_WORD
          ( WORD : LEXICAL_ANALYZER.RESERVED_WORD_KIND ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  if TOKEN.KIND /= LEXICAL_ANALYZER.RESERVED_WORD or else
   TOKEN.RESERVED_WORD /= WORD then
    declare
      IMAGE : constant STRING :=
       LEXICAL_ANALYZER.RESERVED_WORD_KIND'IMAGE ( WORD );
    begin
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
      ( TOKEN , "Expecting " & IMAGE ( 3 .. IMAGE'LAST ) );
    end;
  end if;
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
end PROCESS_RESERVED_WORD;

procedure SKIP_SELECT_CLAUSE is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
  loop
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  exit when TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
   TOKEN.ID.all = "FROM";
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
     TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
      ( TOKEN , "Missing FROM clause" );
    end if;
    LEXICAL_ANALYZER.SKIP_TOKEN_FOR_NOW;
  end loop;
end SKIP_SELECT_CLAUSE;

end SYNTACTICALLY;
```

3.11.70 package tents.ada

```
-- tents.ada - internal data structure for the tentative function list
```

```
with ADA_SQL_FUNCTION_DEFINITIONS, CORRELATION, DDL_DEFINITIONS, RESULT,
     SELEC;
package TENTATIVE is

-- As we parse and process Ada/SQL statements, we come across various
-- functions that we must generate.  Unfortunately, when we see that we must
-- generate a function, we may not know all that we need to know about it.
-- In particular, we may not know (1) whether a strongly typed return is
-- required in the context, or whether the function should return SQL_OBJECT,
-- and (2) precisely what the types of the function's parameter(s) and return
-- are anyway.

-- Example of (1): Suppose we are looking at the column name in the following
-- two examples:
--
--   SELEC ( COLUMN , ...
--
--   SELEC ( COLUMN + 2 , ...
--
-- We know that we must generate a column name function, but the first example
-- requires an SQL_OBJECT return, while the second example requires a strongly
-- typed return.

-- Example of (2):
--
--   SELEC ( INDICATOR ( 2 ) + COLUMN , ...
--
-- When we process the INDICATOR function, we do not know what the type of "2"
-- is.  (A contrived example, admittedly, BUT WE HANDLE IT!)

-- When we see that we must generate a function, but do not yet know
-- everything about it, we put a description of the function (as much as we
-- know) on a tentative function list.  Each routine that processes an
-- expression-type construct builds a tentative function list, and returns it
-- to its caller.  The caller then decides what to do with the list -- if it
-- has more type information than the routine that built the list, then it
-- will nail down all the functions on the list, causing them to be generated.
-- If it still thinks that the functions are tentative, then it will pass the
-- list (perhaps augmented with its own operation) on to its caller, and so
-- on.  The routines in this package are concerned with building and
-- manipulating tentative function lists.

-- Example: Processing INDICATOR ( 2 ) + COLUMN
--
-- (1) The routine processing INDICATOR returns a tentative function list (A),
--     noting that an INDICATOR function is required.  The exact types of
--     INDICATOR's parameter and return are not yet known; they are some
--     integer type.  (The routine processing INDICATOR also calls a routine
--     that processes "2", but we'll forget about that for the example.)
```

```
--
-- (2) The routine processing COLUMN returns a tentative function list (B),
--     noting that a function for the COLUMN name must be generated.  Since
--     database columns have specific types, the return type of COLUMN is
--     known.
--
-- (3) Both (1) and (2) have been called by the routine processing +.  It
--     looks at the types of the operands and, since the type of the right
--     operand is fully known, decides that that determines the type of the
--     left operand.  It causes the functions of tentative function lists (A)
--     (INDICATOR) and (B) (COLUMN) to be flagged as requiring generation with
--     strongly typed returns, since + is a strongly typed operation.  The +
--     routine then builds its own tentative function list to return to its
--     caller, noting that a + function must be generated.  The parameter and
--     return types are all the same -- the strongly typed database type of
--     COLUMN.  But the + function is still tentative because the return type
--     will be changed to SQL_OBJECT if the result of the + does not require
--     strong typing (e.g., is used as an element in a select list).

-- The information that we have to know about a function to be generated
-- obviously differs depending on the kind of function it is.  In all cases,
-- however, we need to know the types of the parameters and the return.  The
-- RESULT.DESCRIPTOR data structure (see results.ada) describes our current
-- state of knowledge about a type required as a function parameter or return
-- result.

-- Associated with a parameter or return type is an action.  In our + example
-- above, the parameter types for + will always be generated as is -- strongly
-- typed.  The return type of +, however, may be replaced with SQL_OBJECT,
-- which would be the type determined as required by an outer routine.  When
-- an entry is made to a tentative function list, actions are specified for
-- all parameter and return types.  The two actions are designated by values
-- of the TENTATIVE.TYPE_ACTION enumeration type:

  type TYPE_ACTION is ( TYPE_MUST_BE_USED_AS_IS , TYPE_MAY_BE_REPLACED );

-- As already noted, every tentative function requires information about its
-- return type and return action.  Other information required differs
-- according to the kind of function, as described below according to values
-- of an enumeration type descriptive of the kinds of tentative functions:

-- UNARY_OPERATION
-- (1) Kind of operation (e.g., AVG)
-- (2) Parameter type
-- (3) Parameter action

-- BINARY_OPERATION
-- (1) Kind of operation (e.g., +)
-- (2) Left parameter type
```

```
-- (3) Left parameter action
-- (4) Right parameter type
-- (5) Right parameter action

-- UNQUALIFIED_COLUMN_NAME
-- (1) Name of the column

-- QUALIFIED_COLUMN_NAME (table.column, see below for correlation_name.column)
-- (1) Identity of the column

-- CORRELATED_COLUMN_NAME (correlation_name.column)
-- (1) Identity of the correlation name
-- (2) Identity of the column

-- CONVERT_TO_FUNCTION
-- (1) No additional information required - the return type defines the
--     function

-- INDICATOR_FUNCTION
-- (1) No additional information required - the return type defines the
--     function

-- COUNT_STAR
-- (1) No additional information required - the return type is always based
--     on DATABASE.INT; COUNT ( '*' ) is put on the tentative function list
--     because there are contexts where it should return SQL_OBJECT instead of
--     a type based on DATABASE.INT

-- SELECT_FUNCTION
-- (1) Routine name - see SELEC.ROUTINE_NAME in selecs.ada
-- (2) Parameter kind - see SELEC.PARAMETER_TYPE - only subqueries wind up in
--     a tentative function list, since they may involve strong typing;
--     subprograms for the other types of selects are generated directly.
--     Consequently, SELEC.SQL_OBJECT is not a possible parameter kind on a
--     tentative function list.
-- (3) Result kind - see SELEC.RESULT_TYPE - this is not actually stored.
--     Since subqueries are the only selects for which the tentative function
--     list is used, possible values of this flag would be SELEC.SQL_OBJECT or
--     SELEC.DATABASE_VALUE.  While on the tentative function list, a select
--     is marked with the tentative function list designation for returning a
--     database value.  When the select function is actually flagged as
--     requiring generation, it may return a database value or an SQL_OBJECT,
--     depending on the context in which it is used.  (SQL_OBJECT return not
--     used in the current implementation.)
-- (4) Parameter type and action (in tentative function list terminology) -
--     not actually stored.  The parameter type can be inferred from the
--     return type (parameter and return are comparable types, unless
--     parameter is '*') and item (2).  The action is always assumed to be
--     TENTATIVE.TYPE_MAY_BE_REPLACED.
```

```
  type FUNCTION_KIND is
       ( UNARY_OPERATION          , BINARY_OPERATION      ,
         UNQUALIFIED_COLUMN_NAME  , QUALIFIED_COLUMN_NAME ,
         CORRELATED_COLUMN_NAME   , CONVERT_TO_FUNCTION   ,
         INDICATOR_FUNCTION       , COUNT_STAR            ,
         SELECT_FUNCTION );

-- The actual data structure for storing all this information is private, and
-- so appears later in this specification.  The items of information are
-- stored in the same order as described above.  The routines of this package
-- use the TENTATIVE.FUNCTION_LIST data structure as parameters and/or return
-- values as they process tentative function lists.  The visible declaration
-- is:

  type FUNCTION_LIST is private;

-- There are four categories of operations defined on tentative function
-- lists:
--
-- (1) Create a new tentative function list
--
-- (2) Add a function to a tentative function list
--
-- (3) Combine two tentative function lists into one list
--
-- (4) Flag the functions on a tentative function list as requiring generation

-- Group 1 functions: Create a new tentative function list
--
-- TENTATIVE.FUNCTION_LIST_CREATOR is called to return a new, empty tentative
-- function list:

  function FUNCTION_LIST_CREATOR return FUNCTION_LIST;

-- Group 2 functions: Add a function to a tentative function list

-- Each kind of function that may be represented in a tentative function list
-- has its own procedure for placing a function on a list.  The first
-- parameter to each procedure is the tentative function list on which to
-- place the new function.  The second parameter is the return type of the
-- function to be generated, and the last parameter is the return action,
-- defaulting to TENTATIVE.TYPE_MAY_BE_REPLACED.  The intervening parameters
-- represent the items of information required for each kind of function, in
-- the order discussed above, except that all action information is gathered
-- as the last parameters, and given defaults that I think may represent the
-- only way the routines would be called with our logic.  (I wasn't daring
-- enough to totally omit the parameters, however.)  In some cases the return
-- type parameter can actually be derived from other information -- I have
-- marked those cases, but have left a return type parameter to the procedures
```

```
-- just for the sake of uniformity.  The procedures are:

  procedure FUNCTION_REQUIRED_FOR_UNARY_OPERATION
            ( LIST              : in out FUNCTION_LIST;
              RETURN_TYPE       : in     RESULT.DESCRIPTOR;
              UNARY_OPERATOR    : in     ADA_SQL_FUNCTION_DEFINITIONS.
                                          SQL_OPERATION;
              PARAMETER_TYPE    : in     RESULT.DESCRIPTOR;
              PARAMETER_ACTION  : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED;
              RETURN_ACTION     : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_BINARY_OPERATION
            ( LIST                   : in out FUNCTION_LIST;
              RETURN_TYPE            : in     RESULT.DESCRIPTOR;
              BINARY_OPERATOR        : in     ADA_SQL_FUNCTION_DEFINITIONS.
                                               SQL_OPERATION;
              LEFT_PARAMETER_TYPE    : in     RESULT.DESCRIPTOR;
              RIGHT_PARAMETER_TYPE   : in     RESULT.DESCRIPTOR;
              LEFT_PARAMETER_ACTION  : in     TYPE_ACTION :=
                                               TYPE_MUST_BE_USED_AS_IS;
              RIGHT_PARAMETER_ACTION : in     TYPE_ACTION :=
                                               TYPE_MUST_BE_USED_AS_IS;
              RETURN_ACTION          : in     TYPE_ACTION :=
                                               TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_UNQUALIFIED_COLUMN_NAME
            ( LIST                : in out FUNCTION_LIST;
              RETURN_TYPE         : in     RESULT.DESCRIPTOR;
              UNQUALIFIED_COLUMN  : in     DDL_DEFINITIONS.TYPE_NAME;
              RETURN_ACTION       : in     TYPE_ACTION :=
                                            TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_QUALIFIED_COLUMN_NAME
            ( LIST              : in out FUNCTION_LIST;
              RETURN_TYPE       : in     RESULT.DESCRIPTOR; -- redundant here
              QUALIFIED_COLUMN  : in     DDL_DEFINITIONS.
                                          ACCESS_FULL_NAME_DESCRIPTOR;
              RETURN_ACTION     : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_CORRELATED_COLUMN_NAME
            ( LIST              : in out FUNCTION_LIST;
              RETURN_TYPE       : in     RESULT.DESCRIPTOR; -- redundant here
              CORRELATION_NAME  : in     CORRELATION.NAME_DECLARED_ENTRY;
              COLUMN_NAME       : in     DDL_DEFINITIONS.
                                          ACCESS_FULL_NAME_DESCRIPTOR;
              RETURN_ACTION     : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_CONVERT_TO_FUNCTION
            ( LIST           : in out FUNCTION_LIST;
```

```
                 RETURN_TYPE   : in     RESULT.DESCRIPTOR;
                 RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_INDICATOR_FUNCTION
               ( LIST          : in out FUNCTION_LIST;
                 RETURN_TYPE   : in     RESULT.DESCRIPTOR;
                 RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_COUNT_STAR
               ( LIST          : in out FUNCTION_LIST;
                 RETURN_TYPE   : in     RESULT.DESCRIPTOR; -- unnecessary here
                 RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

  procedure FUNCTION_REQUIRED_FOR_SELECT_FUNCTION
               ( LIST           : in out FUNCTION_LIST;
                 RETURN_TYPE    : in     RESULT.DESCRIPTOR;
                 ROUTINE_NAME   : in     SELEC.ROUTINE_NAME;
                 PARAMETER_KIND : in     SELEC.PARAMETER_TYPE;
                 RETURN_ACTION  : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED );

-- Group 3 functions: Combine two tentative function lists into one list

-- When a binary operator with operands of unknown type is processed, the
-- tentative function lists for the two operands are merged into a single list
-- to be returned for the binary operator.  Since the binary operator is
-- strongly typed, the return actions of all functions on the merged tentative
-- function list are set to TENTATIVE.TYPE_MUST_BE_USED_AS_IS.  TENTATIVE.-
-- FUNCTION_LIST_MERGE performs this function, returning the merge of its two
-- operands:

  function FUNCTION_LIST_MERGE ( A , B : FUNCTION_LIST ) return FUNCTION_LIST;

-- Group 4 functions: Flag the functions on a tentative function list as
-- requiring generation

-- There are two possibilities for flagging functions on a tentative function
-- list as requiring generation: (1) they can be set to return strongly
-- typed, or (2) they can be set to return SQL_OBJECT.

-- When functions are flagged to return strongly typed, any unknown types are
-- set to the appropriate analogue (program or database) of the given type.
-- TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED flags functions on the given
-- tentative function list as returning strongly typed:

  procedure FUNCTIONS_RETURN_STRONGLY_TYPED
               ( LIST        : FUNCTION_LIST;
                 STRONG_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR );

-- When functions are flagged to return SQL_OBJECT, any unknown types are set
```

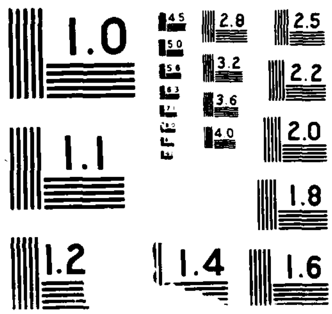
```
-- to the appropriate analogue (program or database) of the STANDARD type of
-- the same class (INTEGER, FLOAT, or STRING; our program logic should prevent
-- us from trying to generate a function for an unknown enumeration type).
-- TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT flags functions on the given
-- tentative function list as returning SQL_OBJECT:

  procedure FUNCTIONS_RETURN_SQL_OBJECT ( LIST : FUNCTION_LIST );

private

  type FUNCTION_LIST_RECORD ( KIND : FUNCTION_KIND );

  type FUNCTION_LIST is access FUNCTION_LIST_RECORD;

  type FUNCTION_LIST_RECORD ( KIND : FUNCTION_KIND ) is
    record
      NEXT_FUNCTION : FUNCTION_LIST;
      RETURN_TYPE   : RESULT.DESCRIPTOR;
      RETURN_ACTION : TYPE_ACTION;
      case KIND is
        when UNARY_OPERATION =>
          UNARY_OPERATOR   : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
          PARAMETER_TYPE   : RESULT.DESCRIPTOR;
          PARAMETER_ACTION : TYPE_ACTION;
        when BINARY_OPERATION =>
          BINARY_OPERATOR        : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
          LEFT_PARAMETER_TYPE    : RESULT.DESCRIPTOR;
          LEFT_PARAMETER_ACTION  : TYPE_ACTION;
          RIGHT_PARAMETER_TYPE   : RESULT.DESCRIPTOR;
          RIGHT_PARAMETER_ACTION : TYPE_ACTION;
        when UNQUALIFIED_COLUMN_NAME =>
          UNQUALIFIED_COLUMN : DDL_DEFINITIONS.TYPE_NAME;
        when QUALIFIED_COLUMN_NAME =>
          QUALIFIED_COLUMN : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when CORRELATED_COLUMN_NAME =>
          CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY;
          COLUMN_NAME      : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when CONVERT_TO_FUNCTION | INDICATOR_FUNCTION | COUNT_STAR =>
          null;
        when SELECT_FUNCTION =>
          ROUTINE_NAME   : SELEC.ROUTINE_NAME;
          PARAMETER_KIND : SELEC.PARAMETER_TYPE;
      end case;
    end record;

end TENTATIVE;
```

3.11.71 package tentb.ada

```
-- tentb.ada - internal data structure for the tentative function list
```

AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317 4/6

UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

```
with CONVERT_TO, GENERATED_FUNCTIONS, INDICATOR, PREDEFINED, RESULT,
 QUALIFIED_NAME, SEMANTICALLY, UNQUALIFIED_NAME;
 use RESULT;
package body TENTATIVE is

   use DDL_DEFINITIONS;

TYPED_OPERAND_KIND : array ( RESULT.VALUE_LOCATION )
 of GENERATED_FUNCTIONS.OPERAND_KIND :=
 ( RESULT.IN_PROGRAM  => GENERATED_FUNCTIONS.O_USER_TYPE,
   RESULT.IN_DATABASE => GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT );

UNTYPED_OPERAND_KIND : constant array ( RESULT.VALUE_LOCATION , TYPE_ACTION )
 of GENERATED_FUNCTIONS.OPERAND_KIND :=
( RESULT.IN_PROGRAM =>
  ( TYPE_MUST_BE_USED_AS_IS => GENERATED_FUNCTIONS.O_USER_TYPE,
    TYPE_MAY_BE_REPLACED    => GENERATED_FUNCTIONS.O_USER_TYPE ),
  RESULT.IN_DATABASE =>
  ( TYPE_MUST_BE_USED_AS_IS => GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT,
    TYPE_MAY_BE_REPLACED    => GENERATED_FUNCTIONS.O_SQL_OBJECT ) );

function FUNCTION_LIST_CREATOR
   return FUNCTION_LIST is
begin
   return null;
end FUNCTION_LIST_CREATOR;

procedure FUNCTION_REQUIRED_FOR_UNARY_OPERATION
   (LIST             : in out FUNCTION_LIST;
    RETURN_TYPE      : in     RESULT.DESCRIPTOR;
    UNARY_OPERATOR   : in     ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
    PARAMETER_TYPE   : in     RESULT.DESCRIPTOR;
    PARAMETER_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED;
    RETURN_ACTION    : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND             => UNARY_OPERATION,
              NEXT_FUNCTION    => LIST,
              RETURN_TYPE      => RETURN_TYPE,
              RETURN_ACTION    => RETURN_ACTION,
              UNARY_OPERATOR   => UNARY_OPERATOR,
              PARAMETER_TYPE   => PARAMETER_TYPE,
              PARAMETER_ACTION => PARAMETER_ACTION);
end FUNCTION_REQUIRED_FOR_UNARY_OPERATION;

procedure FUNCTION_REQUIRED_FOR_BINARY_OPERATION
   (LIST                   : in out FUNCTION_LIST;
    RETURN_TYPE            : in     RESULT.DESCRIPTOR;
    BINARY_OPERATOR        : in     ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
```

```
    LEFT_PARAMETER_TYPE    : in      RESULT.DESCRIPTOR;
    RIGHT_PARAMETER_TYPE   : in      RESULT.DESCRIPTOR;
    LEFT_PARAMETER_ACTION  : in      TYPE_ACTION := TYPE_MUST_BE_USED_AS_IS;
    RIGHT_PARAMETER_ACTION : in      TYPE_ACTION := TYPE_MUST_BE_USED_AS_IS;
    RETURN_ACTION          : in      TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND                   => BINARY_OPERATION,
              NEXT_FUNCTION          => LIST,
              RETURN_TYPE            => RETURN_TYPE,
              RETURN_ACTION          => RETURN_ACTION,
              BINARY_OPERATOR        => BINARY_OPERATOR,
              LEFT_PARAMETER_TYPE    => LEFT_PARAMETER_TYPE,
              RIGHT_PARAMETER_TYPE   => RIGHT_PARAMETER_TYPE,
              LEFT_PARAMETER_ACTION  => LEFT_PARAMETER_ACTION,
              RIGHT_PARAMETER_ACTION => RIGHT_PARAMETER_ACTION);
end FUNCTION_REQUIRED_FOR_BINARY_OPERATION;

procedure FUNCTION_REQUIRED_FOR_UNQUALIFIED_COLUMN_NAME
   (LIST                : in out FUNCTION_LIST;
    RETURN_TYPE         : in     RESULT.DESCRIPTOR;
    UNQUALIFIED_COLUMN  : in     DDL_DEFINITIONS.TYPE_NAME;
    RETURN_ACTION       : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND               => UNQUALIFIED_COLUMN_NAME,
              NEXT_FUNCTION      => LIST,
              RETURN_TYPE        => RETURN_TYPE,
              RETURN_ACTION      => RETURN_ACTION,
              UNQUALIFIED_COLUMN => UNQUALIFIED_COLUMN);
end FUNCTION_REQUIRED_FOR_UNQUALIFIED_COLUMN_NAME;

procedure FUNCTION_REQUIRED_FOR_QUALIFIED_COLUMN_NAME
   (LIST              : in out FUNCTION_LIST;
    RETURN_TYPE       : in     RESULT.DESCRIPTOR;
    QUALIFIED_COLUMN  : in     DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    RETURN_ACTION     : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND             => QUALIFIED_COLUMN_NAME,
              NEXT_FUNCTION    => LIST,
              RETURN_TYPE      => RETURN_TYPE,
              RETURN_ACTION    => RETURN_ACTION,
              QUALIFIED_COLUMN => QUALIFIED_COLUMN);
end FUNCTION_REQUIRED_FOR_QUALIFIED_COLUMN_NAME;

procedure FUNCTION_REQUIRED_FOR_CORRELATED_COLUMN_NAME
   (LIST              : in out FUNCTION_LIST;
    RETURN_TYPE       : in     RESULT.DESCRIPTOR;
```

```
    CORRELATION_NAME : in     CORRELATION.NAME_DECLARED_ENTRY;
    COLUMN_NAME      : in     DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    RETURN_ACTION    : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND             => CORRELATED_COLUMN_NAME,
              NEXT_FUNCTION    => LIST,
              RETURN_TYPE      => RETURN_TYPE,
              RETURN_ACTION    => RETURN_ACTION,
              CORRELATION_NAME => CORRELATION_NAME,
              COLUMN_NAME      => COLUMN_NAME);
end FUNCTION_REQUIRED_FOR_CORRELATED_COLUMN_NAME;

procedure FUNCTION_REQUIRED_FOR_CONVERT_TO_FUNCTION
   (LIST          : in out FUNCTION_LIST;
    RETURN_TYPE   : in     RESULT.DESCRIPTOR;
    RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND          => CONVERT_TO_FUNCTION,
              NEXT_FUNCTION => LIST,
              RETURN_TYPE   => RETURN_TYPE,
              RETURN_ACTION => RETURN_ACTION);
end FUNCTION_REQUIRED_FOR_CONVERT_TO_FUNCTION;

procedure FUNCTION_REQUIRED_FOR_INDICATOR_FUNCTION
   (LIST          : in out FUNCTION_LIST;
    RETURN_TYPE   : in     RESULT.DESCRIPTOR;
    RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND          => INDICATOR_FUNCTION,
              NEXT_FUNCTION => LIST,
              RETURN_TYPE   => RETURN_TYPE,
              RETURN_ACTION => RETURN_ACTION);
end FUNCTION_REQUIRED_FOR_INDICATOR_FUNCTION;

procedure FUNCTION_REQUIRED_FOR_COUNT_STAR
   (LIST          : in out FUNCTION_LIST;
    RETURN_TYPE   : in     RESULT.DESCRIPTOR;
    RETURN_ACTION : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND          => COUNT_STAR,
              NEXT_FUNCTION => LIST,
              RETURN_TYPE   => RETURN_TYPE,
              RETURN_ACTION => RETURN_ACTION);
end FUNCTION_REQUIRED_FOR_COUNT_STAR;
```

```
procedure FUNCTION_REQUIRED_FOR_SELECT_FUNCTION
   (LIST            : in out FUNCTION_LIST;
    RETURN_TYPE     : in     RESULT.DESCRIPTOR;
    ROUTINE_NAME    : in     SELEC.ROUTINE_NAME;
    PARAMETER_KIND  : in     SELEC.PARAMETER_TYPE;
    RETURN_ACTION   : in     TYPE_ACTION := TYPE_MAY_BE_REPLACED) is
begin
   LIST := new FUNCTION_LIST_RECORD'
             (KIND           => SELECT_FUNCTION,
              NEXT_FUNCTION  => LIST,
              RETURN_TYPE    => RETURN_TYPE,
              RETURN_ACTION  => RETURN_ACTION,
              ROUTINE_NAME   => ROUTINE_NAME,
              PARAMETER_KIND => PARAMETER_KIND);
end FUNCTION_REQUIRED_FOR_SELECT_FUNCTION;

function FUNCTION_LIST_MERGE
   (A, B : FUNCTION_LIST)
   return FUNCTION_LIST is
   TRACER : FUNCTION_LIST;
   RESULT : FUNCTION_LIST;
begin
   if A = null then
      RESULT := B;
   elsif B = null then
      RESULT := A;
   else
      -- Add B to end of A's list.
      TRACER := A;
      while TRACER.NEXT_FUNCTION /= null loop
         TRACER := TRACER.NEXT_FUNCTION;
      end loop;
      TRACER.NEXT_FUNCTION := B;
      RESULT := A;
   end if;
   TRACER := RESULT;
   while TRACER /= null loop
      TRACER.RETURN_ACTION := TENTATIVE.TYPE_MUST_BE_USED_AS_IS;
      TRACER := TRACER.NEXT_FUNCTION;
   end loop;
   return RESULT;
end FUNCTION_LIST_MERGE;

procedure FUNCTIONS_RETURN_STRONGLY_TYPED
   (LIST        : FUNCTION_LIST;
    STRONG_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR) is
  TRACER : FUNCTION_LIST := LIST;
begin
   while TRACER /= null loop
```

```
        case TRACER.KIND is
           when UNARY_OPERATION =>
              GENERATED_FUNCTIONS.ADD_UNARY_FUNCTION
                 (TRACER.UNARY_OPERATOR,
                  TYPED_OPERAND_KIND (TRACER.PARAMETER_TYPE.LOCATION),
                  STRONG_TYPE.FULL_NAME,
                  GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT,
                  STRONG_TYPE.FULL_NAME);
           when BINARY_OPERATION =>
              GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
                 (TRACER.BINARY_OPERATOR,
                  TYPED_OPERAND_KIND (TRACER.LEFT_PARAMETER_TYPE.LOCATION),
                  STRONG_TYPE.FULL_NAME,
                  TYPED_OPERAND_KIND (TRACER.RIGHT_PARAMETER_TYPE.LOCATION),
                  STRONG_TYPE.FULL_NAME,
                  GENERATED_FUNCTIONS.O_TYPED_SQL_OBJECT,
                  STRONG_TYPE.FULL_NAME);
           when UNQUALIFIED_COLUMN_NAME =>
              UNQUALIFIED_NAME.RETURNS_TYPED_RESULT
                (TRACER.UNQUALIFIED_COLUMN, STRONG_TYPE.FULL_NAME);
           when QUALIFIED_COLUMN_NAME =>
              QUALIFIED_NAME.RETURNS_STRONGLY_TYPED (TRACER.QUALIFIED_COLUMN);
           when CORRELATED_COLUMN_NAME =>
              CORRELATION.COLUMN_RETURNS_STRONGLY_TYPED
                 (TRACER.CORRELATION_NAME, TRACER.COLUMN_NAME);
           when CONVERT_TO_FUNCTION =>
              CONVERT_TO.RETURNS_STRONGLY_TYPED (STRONG_TYPE.FULL_NAME);
           when INDICATOR_FUNCTION =>
              INDICATOR.RETURNS_STRONGLY_TYPED (STRONG_TYPE.FULL_NAME);
           when COUNT_STAR =>
              PREDEFINED.TEXT_REQUIRED_FOR
              (PREDEFINED.TYPED_COUNT_STAR_FUNCTION);
           when SELECT_FUNCTION =>
              SELEC.REQUIRED_FOR
                 (TRACER.ROUTINE_NAME,
                  TRACER.PARAMETER_KIND,
                  SELEC.DATABASE_VALUE,
                  STRONG_TYPE.FULL_NAME);
        end case;
        TRACER := TRACER.NEXT_FUNCTION;
     end loop;
  end FUNCTIONS_RETURN_STRONGLY_TYPED;

  function STRONGLY_TYPE ( RETURN_TYPE      : RESULT.DESCRIPTOR;
                           PARAMETER_TYPE   : RESULT.DESCRIPTOR;
                           PARAMETER_ACTION : TYPE_ACTION )
   return DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR is
  begin
    if PARAMETER_TYPE.LOCATION = RESULT.IN_DATABASE and then
```

```
     PARAMETER_ACTION = TYPE_MAY_BE_REPLACED then
      return null;
    end if;
    return SEMANTICALLY.STRONGLY_TYPE ( RETURN_TYPE ) . FULL_NAME;
end STRONGLY_TYPE;

procedure FUNCTIONS_RETURN_SQL_OBJECT (LIST : FUNCTION_LIST) is
   TRACER : FUNCTION_LIST := LIST;
begin
     while TRACER /= null loop
         case TRACER.KIND is
             when UNARY_OPERATION =>
                 GENERATED_FUNCTIONS.ADD_UNARY_FUNCTION
                     (TRACER.UNARY_OPERATOR,
                      UNTYPED_OPERAND_KIND
                      (TRACER.PARAMETER_TYPE.LOCATION,TRACER.PARAMETER_ACTION),
                      STRONGLY_TYPE
                      ( TRACER.RETURN_TYPE , TRACER.PARAMETER_TYPE ,
                        TRACER.PARAMETER_ACTION ),
                      GENERATED_FUNCTIONS.O_SQL_OBJECT,
                      null );
             when BINARY_OPERATION =>
                 GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
                     (TRACER.BINARY_OPERATOR,
                      UNTYPED_OPERAND_KIND
                      (TRACER.LEFT_PARAMETER_TYPE.LOCATION,
                       TRACER.LEFT_PARAMETER_ACTION),
                      STRONGLY_TYPE
                      ( TRACER.RETURN_TYPE , TRACER.LEFT_PARAMETER_TYPE ,
                        TRACER.LEFT_PARAMETER_ACTION ),
                      UNTYPED_OPERAND_KIND
                      (TRACER.RIGHT_PARAMETER_TYPE.LOCATION,
                       TRACER.RIGHT_PARAMETER_ACTION),
                      STRONGLY_TYPE
                      ( TRACER.RETURN_TYPE , TRACER.RIGHT_PARAMETER_TYPE ,
                        TRACER.RIGHT_PARAMETER_ACTION ),
                      GENERATED_FUNCTIONS.O_SQL_OBJECT,
                      null );
             when UNQUALIFIED_COLUMN_NAME =>
                 UNQUALIFIED_NAME.RETURNS_SQL_OBJECT(TRACER.UNQUALIFIED_COLUMN);
             when QUALIFIED_COLUMN_NAME =>
                 QUALIFIED_NAME.RETURNS_SQL_OBJECT (TRACER.QUALIFIED_COLUMN);
             when CORRELATED_COLUMN_NAME =>
                 CORRELATION.COLUMN_RETURNS_SQL_OBJECT
                     (TRACER.CORRELATION_NAME, TRACER.COLUMN_NAME);
             when CONVERT_TO_FUNCTION =>
                 CONVERT_TO.RETURNS_SQL_OBJECT
                 (TRACER.RETURN_TYPE.KNOWN_TYPE.FULL_NAME);
             when INDICATOR_FUNCTION =>
```

```
            INDICATOR.RETURNS_SQL_OBJECT
            (TRACER.RETURN_TYPE.KNOWN_TYPE.FULL_NAME);
         when COUNT_STAR =>
            PREDEFINED.TEXT_REQUIRED_FOR
            (PREDEFINED.UNTYPED_COUNT_STAR_FUNCTION);
         when SELECT_FUNCTION =>
            SELEC.REQUIRED_FOR
               (TRACER.ROUTINE_NAME,TRACER.PARAMETER_KIND,SELEC.SQL_OBJECT);
      end case;
      TRACER := TRACER.NEXT_FUNCTION;
   end loop;
end FUNCTIONS_RETURN_SQL_OBJECT;

end TENTATIVE;
```

3.11.72 package exprs.ada

```
-- exprs.ada - routines to process expression-type constructs

with FROM_CLAUSE , RESULT , SEMANTICALLY, TENTATIVE;
package EXPRESSION is

  procedure PROCESS_COLUMN_SPECIFICATION
            ( FROM                    : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY         : in  BOOLEAN;
              ALLOW_TYPE_CONVERSION   : in  BOOLEAN;
              TENTATIVE_FUNCTIONS     : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE             : out RESULT.DESCRIPTOR );

  procedure PROCESS_VALUE_EXPRESSION
            ( FROM                  : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY       : in  BOOLEAN;
              LOCATION              : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS   : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE           : out RESULT.DESCRIPTOR;
              SAW_DATABASE_VALUE    : out BOOLEAN );

  procedure PROCESS_VALUE_EXPRESSION
            ( FROM                  : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY       : in  BOOLEAN;
              LOCATION              : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS   : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE           : out RESULT.DESCRIPTOR );

end EXPRESSION;
```

3.11.73 package exprb.ada

```
-- exprb.ada - routines to process expression-type constructs
```

```
with ADA_SQL_FUNCTION_DEFINITIONS, DDL_DEFINITIONS, ENUMERATION,
 LEXICAL_ANALYZER, NAME, PREDEFINED_TYPE, RESULT, SEMANTICALLY,
 SYNTACTICALLY;
 use DDL_DEFINITIONS, LEXICAL_ANALYZER, NAME, RESULT, SEMANTICALLY;
package body EXPRESSION is

  type CONVERSION_RULE is ( BY_ADA_RULES , BY_SQL_RULES );

  type SQL_PRIMARY_WORDS is
  ( AVG_ALL          , MAX_ALL      , MIN_ALL      , SUM_ALL      ,
    AVG              , MAX          , MIN          , SUM          ,
--  AVG_DISTINCT     , MAX_DISTINCT , MIN_DISTINCT , SUM_DISTINCT , -- distinct
--  COUNT_DISTINCT ,                                                -- not imp.
    COUNT            , INDICATOR );

  type SQL_PRIMARY_OPERATIONS is array ( SQL_PRIMARY_WORDS range <> )
   of ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;

  SQL_PRIMARY_OPERATION : constant SQL_PRIMARY_OPERATIONS :=
  ( AVG_ALL    => ADA_SQL_FUNCTION_DEFINITIONS.O_AVG,
    MAX_ALL    => ADA_SQL_FUNCTION_DEFINITIONS.O_MAX,
    MIN_ALL    => ADA_SQL_FUNCTION_DEFINITIONS.O_MIN,
    SUM_ALL    => ADA_SQL_FUNCTION_DEFINITIONS.O_SUM,
    AVG        => ADA_SQL_FUNCTION_DEFINITIONS.O_AVG,
    MAX        => ADA_SQL_FUNCTION_DEFINITIONS.O_MAX,
    MIN        => ADA_SQL_FUNCTION_DEFINITIONS.O_MIN,
    SUM        => ADA_SQL_FUNCTION_DEFINITIONS.O_SUM );

  type SQL_OPERATIONS is array ( LEXICAL_ANALYZER.DELIMITER_KIND ) of
   ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;

  UNARY_SQL_OPERATION : constant SQL_OPERATIONS :=
  ( ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- AMPERSAND
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- APOSTROPHE
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- LEFT_PARENTHESIS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- RIGHT_PARENTHESIS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- STAR
    ADA_SQL_FUNCTION_DEFINITIONS.O_UNARY_PLUS,   -- PLUS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- COMMA
    ADA_SQL_FUNCTION_DEFINITIONS.O_UNARY_MINUS,  -- HYPHEN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- DOT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- SLASH
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- COLON
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- SEMICOLON
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- LESS_THAN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- EQUAL
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- GREATER_THAN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- VERTICAL_BAR
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- ARROW
```

```
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- DOUBLE_DOT
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- DOUBLE_STAR
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- ASSIGNMENT
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- INEQUALITY
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- GREATER_THAN_OR_EQUAL
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- LESS_THAN_OR_EQUAL
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- LEFT_LABEL_BRACKET
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,      -- RIGHT_LABEL_BRACKET
  ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP );    -- BOX

LOCATION_RESTRICTIONS : constant array ( SEMANTICALLY.LOCATION_RESTRICTION )
 of NAME.KIND_RESTRICTION :=
   ( SEMANTICALLY.ADA_VALUE     => NAME.IS_PROGRAM_VALUE,
     SEMANTICALLY.PROGRAM_VALUE => NAME.IS_PROGRAM_VALUE,
     SEMANTICALLY.ANY_VALUE     => NAME.IS_ANYTHING );

function IS_NUMERIC ( T : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR )
 return BOOLEAN is
begin
  if T.WHICH_TYPE = DDL_DEFINITIONS.INT_EGER or else
   T.WHICH_TYPE = DDL_DEFINITIONS.FL_OAT then
    return TRUE;
  else
    return FALSE;
  end if;
end IS_NUMERIC;

procedure INVALID_CONVERSION is
begin
  LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
  ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
    "Invalid conversion" );
end INVALID_CONVERSION;

procedure VALIDATE_ENUMERATION_IS_CONVERTIBLE
          ( TO   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            FROM : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR ) is
begin
  if TO.WHICH_TYPE /= DDL_DEFINITIONS.ENUMERATION then
    INVALID_CONVERSION;
  elsif TO.ULT_PARENT_TYPE /= FROM.ULT_PARENT_TYPE then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
      "Conversion of enumeration types requires relation by derivation" );
  end if;
end VALIDATE_ENUMERATION_IS_CONVERTIBLE;

procedure VALIDATE_NUMERIC_IS_CONVERTIBLE
          ( TO : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR ) is
```

```
begin
  if not IS_NUMERIC ( TO ) then
    INVALID_CONVERSION;
  end if;
end VALIDATE_NUMERIC_IS_CONVERTIBLE;

procedure PREVALIDATE_STRING_IS_CONVERTIBLE
          ( TO : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR ) is
begin
  if TO.WHICH_TYPE /= DDL_DEFINITIONS.STR_ING then
    INVALID_CONVERSION;
  end if;
end PREVALIDATE_STRING_IS_CONVERTIBLE;

procedure VALIDATE_KNOWN_STRING_IS_CONVERTIBLE
          ( TO   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            FROM : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            RULE : CONVERSION_RULE ) is
begin
  PREVALIDATE_STRING_IS_CONVERTIBLE ( TO );
  if RULE = BY_ADA_RULES then -- we know string has single integer index
    if TO.ARRAY_TYPE.BASE_TYPE /= FROM.ARRAY_TYPE.BASE_TYPE then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
        "Ada string conversion requires same component type" );
    end if;
  end if;
end VALIDATE_KNOWN_STRING_IS_CONVERTIBLE;

procedure VALIDATE_UNKNOWN_STRING_IS_CONVERTIBLE
          ( TO   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            RULE : CONVERSION_RULE ) is
begin
  PREVALIDATE_STRING_IS_CONVERTIBLE ( TO );
  if RULE = BY_ADA_RULES then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
      "Ada type conversion cannot be used on string literal" );
  end if;
end VALIDATE_UNKNOWN_STRING_IS_CONVERTIBLE;

procedure VALIDATE_IS_CONVERTIBLE
          ( TO   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            FROM : RESULT.DESCRIPTOR;
            RULE : CONVERSION_RULE ) is
  procedure REPORT_ERROR is
  begin
    LEXICAL_ANALYZER.REPORT_SYSTEM_ERROR
    ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
```

```
        "EXPRESSION.VALIDATE_IS_CONVERTIBLE" );
  end REPORT_ERROR;
begin
  if FROM.TYPE_IS = RESULT.IS_KNOWN then
    case FROM.KNOWN_TYPE.WHICH_TYPE is
      when DDL_DEFINITIONS.INT_EGER | DDL_DEFINITIONS.FL_OAT =>
        VALIDATE_NUMERIC_IS_CONVERTIBLE ( TO );
      when DDL_DEFINITIONS.STR_ING =>
        VALIDATE_KNOWN_STRING_IS_CONVERTIBLE ( TO, FROM.KNOWN_TYPE, RULE );
      when DDL_DEFINITIONS.ENUMERATION =>
        VALIDATE_ENUMERATION_IS_CONVERTIBLE ( TO , FROM.KNOWN_TYPE );
      when others =>
        REPORT_ERROR;
    end case;
  else
    case FROM.UNKNOWN_TYPE.CLASS is
      when DDL_DEFINITIONS.INT_EGER | DDL_DEFINITIONS.FL_OAT =>
        VALIDATE_NUMERIC_IS_CONVERTIBLE ( TO );
      when DDL_DEFINITIONS.STR_ING =>
        VALIDATE_UNKNOWN_STRING_IS_CONVERTIBLE ( TO , RULE );
      when DDL_DEFINITIONS.ENUMERATION =>
        LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
        ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
          "Cannot convert from an unknown enumeration type" );
      when others =>
        REPORT_ERROR;
    end case;
  end if;
end VALIDATE_IS_CONVERTIBLE;

procedure VALIDATE_CONVERT_TO
          ( CONVERT_TO_NAME : NAME.INFORMATION;
            PARAMETER       : RESULT.DESCRIPTOR ) is
begin
  if PARAMETER.LOCATION = RESULT.IN_PROGRAM then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
      "Use Ada type conversion for Ada values" );
  end if;
  VALIDATE_IS_CONVERTIBLE
  ( CONVERT_TO_NAME.CONVERT_TO_TYPE.TYPE_IS , PARAMETER , BY_SQL_RULES );
end VALIDATE_CONVERT_TO;

procedure SET_TENTATIVE_FUNCTIONS_FOR_CONVERT_FUNCTION_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  OUR_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST :=
   TENTATIVE.FUNCTION_LIST_CREATOR;
```

```
    OUR_RETURN_TYPE : RESULT.DESCRIPTOR :=
    ( TYPE_IS    => RESULT.IS_KNOWN,
      LOCATION   => RESULT.IN_DATABASE,
      KNOWN_TYPE => NAME_INFORMATION.CONVERT_TO_TYPE.TYPE_IS );
  begin
    TENTATIVE.FUNCTION_REQUIRED_FOR_CONVERT_TO_FUNCTION
    ( OUR_TENTATIVE_FUNCTIONS , OUR_RETURN_TYPE );
    TENTATIVE_FUNCTIONS := OUR_TENTATIVE_FUNCTIONS;
    RETURN_TYPE := OUR_RETURN_TYPE;
  end SET_TENTATIVE_FUNCTIONS_FOR_CONVERT_FUNCTION_NAME;

  procedure SET_TENTATIVE_FUNCTIONS_FOR_CORRELATED_COLUMN_NAME
            ( NAME_INFORMATION    : in  NAME.INFORMATION;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
    OUR_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST :=
     TENTATIVE.FUNCTION_LIST_CREATOR;
    OUR_RETURN_TYPE : RESULT.DESCRIPTOR :=
    ( TYPE_IS    => RESULT.IS_KNOWN,
      LOCATION   => RESULT.IN_DATABASE,
      KNOWN_TYPE => NAME_INFORMATION.CORRELATED_COLUMN.TYPE_IS.BASE_TYPE );
  begin
    TENTATIVE.FUNCTION_REQUIRED_FOR_CORRELATED_COLUMN_NAME
    ( OUR_TENTATIVE_FUNCTIONS,
      OUR_RETURN_TYPE,
      NAME_INFORMATION.CORRELATION_NAME,
      NAME_INFORMATION.CORRELATED_COLUMN );
    TENTATIVE_FUNCTIONS := OUR_TENTATIVE_FUNCTIONS;
    RETURN_TYPE := OUR_RETURN_TYPE;
  end SET_TENTATIVE_FUNCTIONS_FOR_CORRELATED_COLUMN_NAME;

  procedure SET_TENTATIVE_FUNCTIONS_FOR_ENUMERATION_LITERAL_NAME
            ( NAME_INFORMATION    : in  NAME.INFORMATION;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  begin
    TENTATIVE_FUNCTIONS := TENTATIVE.FUNCTION_LIST_CREATOR;
    case ENUMERATION.TYPE_LIST_SIZE(NAME_INFORMATION.ENUMERATION_TYPE_LIST) is
      when 1 =>
        RETURN_TYPE :=
        ( TYPE_IS    => RESULT.IS_KNOWN,
          LOCATION   => RESULT.IN_PROGRAM,
          KNOWN_TYPE => ENUMERATION.TYPE_ON_LIST
                        ( NAME_INFORMATION.ENUMERATION_TYPE_LIST ).TYPE_IS );
      when 2 =>
        RETURN_TYPE :=
        ( TYPE_IS =>
           RESULT.IS_UNKNOWN,
          LOCATION =>
```

```
        RESULT.IN_PROGRAM,
       UNKNOWN_TYPE =>
       ( CLASS          => DDL_DEFINITIONS.ENUMERATION,
         POSSIBLE_TYPES => NAME_INFORMATION.ENUMERATION_TYPE_LIST ) );
    when others =>
      LEXICAL_ANALYZER.REPORT_SYSTEM_ERROR
      ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
        "EXPRESSION.SET_TENTATIVE_FUNCTIONS_FOR_ENUMERATION_LITERAL_NAME" );
  end case;
end SET_TENTATIVE_FUNCTIONS_FOR_ENUMERATION_LITERAL_NAME;

procedure SET_TENTATIVE_FUNCTIONS_FOR_PROGRAM_TYPE_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
begin
  TENTATIVE_FUNCTIONS := TENTATIVE.FUNCTION_LIST_CREATOR;
  RETURN_TYPE :=
  ( TYPE_IS    => RESULT.IS_KNOWN,
    LOCATION   => RESULT.IN_PROGRAM,
    KNOWN_TYPE => NAME_INFORMATION.PROGRAM_TYPE.TYPE_IS );
end SET_TENTATIVE_FUNCTIONS_FOR_PROGRAM_TYPE_NAME;

procedure SET_TENTATIVE_FUNCTIONS_FOR_QUALIFIED_COLUMN_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  OUR_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST :=
   TENTATIVE.FUNCTION_LIST_CREATOR;
  OUR_RETURN_TYPE : RESULT.DESCRIPTOR :=
  ( TYPE_IS    => RESULT.IS_KNOWN,
    LOCATION   => RESULT.IN_DATABASE,
    KNOWN_TYPE => NAME_INFORMATION.QUALIFIED_COLUMN.TYPE_IS.BASE_TYPE );
begin
  TENTATIVE.FUNCTION_REQUIRED_FOR_QUALIFIED_COLUMN_NAME
  ( OUR_TENTATIVE_FUNCTIONS,
    OUR_RETURN_TYPE,
    NAME_INFORMATION.QUALIFIED_COLUMN );
  TENTATIVE_FUNCTIONS := OUR_TENTATIVE_FUNCTIONS;
  RETURN_TYPE := OUR_RETURN_TYPE;
end SET_TENTATIVE_FUNCTIONS_FOR_QUALIFIED_COLUMN_NAME;

procedure SET_TENTATIVE_FUNCTIONS_FOR_UNQUALIFIED_COLUMN_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  OUR_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST :=
   TENTATIVE.FUNCTION_LIST_CREATOR;
  OUR_RETURN_TYPE : RESULT.DESCRIPTOR :=
```

```
  ( TYPE_IS    => RESULT.IS_KNOWN,
    LOCATION   => RESULT.IN_DATABASE,
    KNOWN_TYPE => NAME_INFORMATION.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE );
begin
  TENTATIVE.FUNCTION_REQUIRED_FOR_UNQUALIFIED_COLUMN_NAME
  ( OUR_TENTATIVE_FUNCTIONS,
    OUR_RETURN_TYPE,
    NAME_INFORMATION.UNQUALIFIED_COLUMN.NAME );
  TENTATIVE_FUNCTIONS := OUR_TENTATIVE_FUNCTIONS;
  RETURN_TYPE := OUR_RETURN_TYPE;
end SET_TENTATIVE_FUNCTIONS_FOR_UNQUALIFIED_COLUMN_NAME;

procedure SET_TENTATIVE_FUNCTIONS_FOR_VARIABLE_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
begin
  TENTATIVE_FUNCTIONS := TENTATIVE.FUNCTION_LIST_CREATOR;
  RETURN_TYPE :=
  ( TYPE_IS    => RESULT.IS_KNOWN,
    LOCATION   => RESULT.IN_PROGRAM,
    KNOWN_TYPE => NAME_INFORMATION.VARIABLE_TYPE.TYPE_IS.BASE_TYPE );
end SET_TENTATIVE_FUNCTIONS_FOR_VARIABLE_NAME;

procedure SET_TENTATIVE_FUNCTIONS_FOR_NAME
          ( NAME_INFORMATION    : in  NAME.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE  : out BOOLEAN ) is
begin
  case NAME_INFORMATION.KIND is
    when NAME.OF_QUALIFIED_COLUMN =>
      SET_TENTATIVE_FUNCTIONS_FOR_QUALIFIED_COLUMN_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := TRUE;
    when NAME.OF_CORRELATED_COLUMN =>
      SET_TENTATIVE_FUNCTIONS_FOR_CORRELATED_COLUMN_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := TRUE;
    when NAME.OF_UNQUALIFIED_COLUMN =>
      SET_TENTATIVE_FUNCTIONS_FOR_UNQUALIFIED_COLUMN_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := TRUE;
    when NAME.OF_CONVERT_FUNCTION =>
      SET_TENTATIVE_FUNCTIONS_FOR_CONVERT_FUNCTION_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when NAME.OF_PROGRAM_TYPE =>
      SET_TENTATIVE_FUNCTIONS_FOR_PROGRAM_TYPE_NAME
```

```
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when NAME.OF_ENUMERATION_LITERAL =>
      SET_TENTATIVE_FUNCTIONS_FOR_ENUMERATION_LITERAL_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when NAME.OF_VARIABLE =>
      SET_TENTATIVE_FUNCTIONS_FOR_VARIABLE_NAME
      ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
  end case;
end SET_TENTATIVE_FUNCTIONS_FOR_NAME;

procedure PROCESS_COLUMN_SPECIFICATION
          ( FROM                   : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY        : in  BOOLEAN;
            ALLOW_TYPE_CONVERSION  : in  BOOLEAN;
            TENTATIVE_FUNCTIONS    : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE            : out RESULT.DESCRIPTOR ) is
  NAME_INFORMATION : NAME.INFORMATION :=
   NAME.AT_CURRENT_INPUT_POINT
   ( SCOPE                 => FROM,
     RESTRICT_SO           => NAME.IS_COLUMN_SPECIFICATION,
     THIS_SCOPE_ONLY       => THIS_SCOPE_ONLY,
     ALLOW_TYPE_CONVERSION => ALLOW_TYPE_CONVERSION );
  PARAMETER_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST;
  PARAMETER_RETURN_TYPE         : RESULT.DESCRIPTOR;
  DUMMY                         : BOOLEAN;
begin
  SYNTACTICALLY.GOBBLE_NAME ( NAME_INFORMATION );
  if NAME_INFORMATION.KIND = NAME.OF_CONVERT_FUNCTION then
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
    PROCESS_COLUMN_SPECIFICATION
    ( FROM                  => FROM,
      THIS_SCOPE_ONLY       => THIS_SCOPE_ONLY,
      ALLOW_TYPE_CONVERSION => TRUE,
      TENTATIVE_FUNCTIONS   => PARAMETER_TENTATIVE_FUNCTIONS,
      RETURN_TYPE           => PARAMETER_RETURN_TYPE );
    VALIDATE_CONVERT_TO ( NAME_INFORMATION , PARAMETER_RETURN_TYPE );
    TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT
    ( PARAMETER_TENTATIVE_FUNCTIONS );
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  end if;
  SET_TENTATIVE_FUNCTIONS_FOR_NAME
  ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE , DUMMY );
end PROCESS_COLUMN_SPECIFICATION;

procedure REPORT_PRIMARY_ERROR ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN ) is
begin
```

```
  LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR ( TOKEN , "Expecting a primary" );
end REPORT_PRIMARY_ERROR;

procedure PROCESS_PRIMARY_CHARACTER_LITERAL
          ( FROM                 : in  FROM_CLAUSE.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  DUMMY : BOOLEAN;
  NAME_INFORMATION : NAME.INFORMATION :=
   NAME.AT_CURRENT_INPUT_POINT
   ( SCOPE                 => FROM,
     RESTRICT_SO           => NAME.IS_PROGRAM_VALUE,
     THIS_SCOPE_ONLY       => TRUE,
     ALLOW_TYPE_CONVERSION => FALSE );
begin
  SYNTACTICALLY.GOBBLE_NAME ( NAME_INFORMATION );
  SET_TENTATIVE_FUNCTIONS_FOR_NAME
  ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE , DUMMY );
end PROCESS_PRIMARY_CHARACTER_LITERAL;

procedure PROCESS_PRIMARY_DELIMITER
          ( TOKEN                : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM                 : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY      : in  BOOLEAN;
            LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE  : out BOOLEAN ) is
begin
  if TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    PROCESS_VALUE_EXPRESSION
    ( FROM , THIS_SCOPE_ONLY , LOCATION , TENTATIVE_FUNCTIONS ,
      RETURN_TYPE , SAW_DATABASE_VALUE );
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  else
    REPORT_PRIMARY_ERROR ( TOKEN );
  end if;
end PROCESS_PRIMARY_DELIMITER;

procedure VALIDATE_IS_QUALIFIABLE
          ( TO   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            FROM : RESULT.DESCRIPTOR ) is
begin
  if FROM.TYPE_IS = RESULT.IS_KNOWN then
    if FROM.KNOWN_TYPE /= TO then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
        "Operand to qualification is not of correct type" );
```

```
      end if;
    else
      if FROM.UNKNOWN_TYPE.CLASS /= TO.WHICH_TYPE then
        LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
        ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
          "Invalid qualification" );
      elsif TO.WHICH_TYPE = DDL_DEFINITIONS.ENUMERATION then
        if not ENUMERATION.TYPE_IS_ON_LIST
               ( TO.FULL_NAME , FROM.UNKNOWN_TYPE.POSSIBLE_TYPES ) then
          LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
          ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN,
            "Enumeration literal not a value of given type" );
        end if;
      end if;
    end if;
  end VALIDATE_IS_QUALIFIABLE;

  procedure PROCESS_NAME_OF_PROGRAM_TYPE
            ( FROM             : FROM_CLAUSE.INFORMATION;
              NAME_INFORMATION : NAME.INFORMATION ) is
    TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
     LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST;
    RETURN_TYPE         : RESULT.DESCRIPTOR;
    procedure REPORT_ERROR is
    begin
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
      ( TOKEN , "Expecting Ada type conversion or qualification" );
    end REPORT_ERROR;
  begin
    if TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER then
      REPORT_ERROR;
    end if;
    case TOKEN.DELIMITER is
      when LEXICAL_ANALYZER.APOSTROPHE =>
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
        PROCESS_VALUE_EXPRESSION
        ( FROM , FALSE , SEMANTICALLY.ADA_VALUE , TENTATIVE_FUNCTIONS ,
          RETURN_TYPE );
        VALIDATE_IS_QUALIFIABLE
        ( NAME_INFORMATION.PROGRAM_TYPE.TYPE_IS, RETURN_TYPE );
      when LEXICAL_ANALYZER.LEFT_PARENTHESIS =>
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        PROCESS_VALUE_EXPRESSION
        ( FROM , FALSE , SEMANTICALLY.ADA_VALUE , TENTATIVE_FUNCTIONS ,
          RETURN_TYPE );
        VALIDATE_IS_CONVERTIBLE
        ( NAME_INFORMATION.PROGRAM_TYPE.TYPE_IS, RETURN_TYPE, BY_ADA_RULES );
```

```
    when others =>
      REPORT_ERROR;
  end case;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
end PROCESS_NAME_OF_PROGRAM_TYPE;

procedure VALIDATE_ADA_SQL_VALUE_ALLOWED
          ( TOKEN    : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            LOCATION : SEMANTICALLY.LOCATION_RESTRICTION ) is
begin
  if LOCATION = SEMANTICALLY.ADA_VALUE then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN , "Only Ada values allowed here" );
  end if;
end VALIDATE_ADA_SQL_VALUE_ALLOWED;

procedure VALIDATE_DATABASE_VALUE_ALLOWED
          ( TOKEN    : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            LOCATION : SEMANTICALLY.LOCATION_RESTRICTION ) is
begin
  if LOCATION /= SEMANTICALLY.ANY_VALUE then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN , "Database value not permitted here" );
  end if;
end VALIDATE_DATABASE_VALUE_ALLOWED;

procedure PROCESS_NAME
          ( TOKEN                : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM                 : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY      : in  BOOLEAN;
            LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS  : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE          : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE   : out BOOLEAN ) is
  NAME_INFORMATION : NAME.INFORMATION :=
   NAME.AT_CURRENT_INPUT_POINT
   ( FROM , LOCATION_RESTRICTIONS ( LOCATION ) ,
     THIS_SCOPE_ONLY , TRUE );
  PARAMETER_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST;
  PARAMETER_RETURN_TYPE         : RESULT.DESCRIPTOR;
  CONVERT_TO_FOR_DATABASE_VALUE : BOOLEAN := FALSE;
  NAME_IS_DATABASE_VALUE        : BOOLEAN;
begin
  SYNTACTICALLY.GOBBLE_NAME ( NAME_INFORMATION );
  case NAME_INFORMATION.KIND is
    when NAME.OF_CONVERT_FUNCTION =>
      SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
      PROCESS_VALUE_EXPRESSION
      ( FROM , THIS_SCOPE_ONLY , SEMANTICALLY.ANY_VALUE ,
```

```
        PARAMETER_TENTATIVE_FUNCTIONS , PARAMETER_RETURN_TYPE ,
        CONVERT_TO_FOR_DATABASE_VALUE );
      VALIDATE_CONVERT_TO ( NAME_INFORMATION , PARAMETER_RETURN_TYPE );
      TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT
      ( PARAMETER_TENTATIVE_FUNCTIONS );
      SYNTACTICALLY.PROCESS_DELIMITER(LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
    when NAME.OF_PROGRAM_TYPE =>
      PROCESS_NAME_OF_PROGRAM_TYPE ( FROM , NAME_INFORMATION );
    when others =>
      null;
  end case;
  SET_TENTATIVE_FUNCTIONS_FOR_NAME
  ( NAME_INFORMATION , TENTATIVE_FUNCTIONS , RETURN_TYPE ,
    NAME_IS_DATABASE_VALUE );
  SAW_DATABASE_VALUE :=
   CONVERT_TO_FOR_DATABASE_VALUE or NAME_IS_DATABASE_VALUE;
end PROCESS_NAME;

procedure VALIDATE_NUMERIC_PARAMETER
          ( TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            RETURN_TYPE : RESULT.DESCRIPTOR ) is
  procedure REPORT_ERROR is
  begin
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN , "Numeric operand(s) required" );
  end REPORT_ERROR;
begin
  case RETURN_TYPE.TYPE_IS is
    when RESULT.IS_KNOWN =>
      if not IS_NUMERIC ( RETURN_TYPE.KNOWN_TYPE ) then
        REPORT_ERROR;
      end if;
    when RESULT.IS_UNKNOWN =>
      case RETURN_TYPE.UNKNOWN_TYPE.CLASS is
        when DDL_DEFINITIONS.INT_EGER | DDL_DEFINITIONS.FL_OAT =>
          null;
        when others =>
          REPORT_ERROR;
      end case;
  end case;
end VALIDATE_NUMERIC_PARAMETER;

procedure PROCESS_ALL_SET_FUNCTION
          ( TOKEN               : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            SQL_KEYWORD         : in  SQL_PRIMARY_WORDS;
            FROM                : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY     : in  BOOLEAN;
            LOCATION            : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
```

```
               RETURN_TYPE           : out RESULT.DESCRIPTOR ) is
    PARAMETER_RETURN_TYPE            : RESULT.DESCRIPTOR;
    PARAMETER_TENTATIVE_FUNCTIONS    : TENTATIVE.FUNCTION_LIST;
    SAW_DATABASE_VALUE               : BOOLEAN;
  begin
    VALIDATE_DATABASE_VALUE_ALLOWED ( TOKEN , LOCATION );
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
    PROCESS_VALUE_EXPRESSION
    ( FROM , THIS_SCOPE_ONLY , SEMANTICALLY.ANY_VALUE ,
      PARAMETER_TENTATIVE_FUNCTIONS , PARAMETER_RETURN_TYPE ,
      SAW_DATABASE_VALUE );
    case SQL_KEYWORD is
      when AVG_ALL | SUM_ALL | AVG | MIN =>
        VALIDATE_NUMERIC_PARAMETER ( TOKEN , PARAMETER_RETURN_TYPE );
      when others =>
        null;
    end case;
    SEMANTICALLY.VALIDATE_DATABASE_VALUE_USED ( TOKEN , SAW_DATABASE_VALUE );
    TENTATIVE.FUNCTION_REQUIRED_FOR_UNARY_OPERATION
    ( LIST           => PARAMETER_TENTATIVE_FUNCTIONS,
      RETURN_TYPE    => PARAMETER_RETURN_TYPE,
      UNARY_OPERATOR => SQL_PRIMARY_OPERATION ( SQL_KEYWORD ),
      PARAMETER_TYPE => PARAMETER_RETURN_TYPE );
    TENTATIVE_FUNCTIONS := PARAMETER_TENTATIVE_FUNCTIONS;
    RETURN_TYPE := PARAMETER_RETURN_TYPE;
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  end PROCESS_ALL_SET_FUNCTION;

  procedure PROCESS_COUNT_STAR
            ( TOKEN                 : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              LOCATION              : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS   : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE           : out RESULT.DESCRIPTOR ) is
    OUR_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
    OUR_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST :=
     TENTATIVE.FUNCTION_LIST_CREATOR;
    OUR_RETURN_TYPE : RESULT.DESCRIPTOR :=
     ( TYPE_IS    => RESULT.IS_KNOWN,
       LOCATION   => RESULT.IN_DATABASE,
       KNOWN_TYPE => PREDEFINED_TYPE.DATABASE.INT );
  begin
    VALIDATE_DATABASE_VALUE_ALLOWED ( TOKEN , LOCATION );
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
    OUR_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if OUR_TOKEN.KIND /= LEXICAL_ANALYZER.CHARACTER_LITERAL or else
     OUR_TOKEN.CHARACTER_VALUE /= '*' then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
```

```
    ( OUR_TOKEN , "Expecting '*' for COUNT ( '*' )" );
  end if;
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  TENTATIVE.FUNCTION_REQUIRED_FOR_COUNT_STAR
  ( OUR_TENTATIVE_FUNCTIONS , OUR_RETURN_TYPE );
  TENTATIVE_FUNCTIONS := OUR_TENTATIVE_FUNCTIONS;
  RETURN_TYPE := OUR_RETURN_TYPE;
end PROCESS_COUNT_STAR;

procedure PROCESS_INDICATOR
          ( TOKEN                : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM                 : in  FROM_CLAUSE.INFORMATION;
            LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS  : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE          : out RESULT.DESCRIPTOR ) is
  PARAMETER_TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST;
  PARAMETER_RETURN_TYPE         : RESULT.DESCRIPTOR;
begin
  VALIDATE_ADA_SQL_VALUE_ALLOWED ( TOKEN , LOCATION );
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
  PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ADA_VALUE , PARAMETER_TENTATIVE_FUNCTIONS ,
    PARAMETER_RETURN_TYPE );
  PARAMETER_RETURN_TYPE.LOCATION := RESULT.IN_DATABASE;
  if
   SEMANTICALLY.VALIDATE_STRONGLY_TYPED ( TOKEN , PARAMETER_RETURN_TYPE ) /=
    null then
    TENTATIVE.FUNCTION_REQUIRED_FOR_INDICATOR_FUNCTION
    ( PARAMETER_TENTATIVE_FUNCTIONS , PARAMETER_RETURN_TYPE );
  end if;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  TENTATIVE_FUNCTIONS := PARAMETER_TENTATIVE_FUNCTIONS;
  RETURN_TYPE := PARAMETER_RETURN_TYPE;
end PROCESS_INDICATOR;

procedure PROCESS_PRIMARY_IDENTIFIER
          ( TOKEN                : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM                 : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY      : in  BOOLEAN;
            LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS  : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE          : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE   : out BOOLEAN ) is
  SQL_KEYWORD : SQL_PRIMARY_WORDS;
begin
  begin
    SQL_KEYWORD := SQL_PRIMARY_WORDS'VALUE ( TOKEN.ID.all );
```

```
    exception
      when CONSTRAINT_ERROR =>
        PROCESS_NAME
        ( TOKEN , FROM , THIS_SCOPE_ONLY , LOCATION , TENTATIVE_FUNCTIONS ,
          RETURN_TYPE , SAW_DATABASE_VALUE );
      return;
    end;
    case SQL_KEYWORD is
      when AVG_ALL | MAX_ALL | MIN_ALL | SUM_ALL | AVG | MAX | MIN | SUM =>
        PROCESS_ALL_SET_FUNCTION
        ( TOKEN , SQL_KEYWORD , FROM , THIS_SCOPE_ONLY , LOCATION ,
          TENTATIVE_FUNCTIONS , RETURN_TYPE );
        SAW_DATABASE_VALUE := TRUE;
      when COUNT =>
        PROCESS_COUNT_STAR
        ( TOKEN , LOCATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
        SAW_DATABASE_VALUE := FALSE;
      when INDICATOR =>
        PROCESS_INDICATOR
        ( TOKEN , FROM , LOCATION , TENTATIVE_FUNCTIONS , RETURN_TYPE );
        SAW_DATABASE_VALUE := FALSE;
    end case;
  end PROCESS_PRIMARY_IDENTIFIER;

  procedure PROCESS_PRIMARY_NUMERIC_LITERAL
            ( TOKEN               : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  begin
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    TENTATIVE_FUNCTIONS := TENTATIVE.FUNCTION_LIST_CREATOR;
    if SYNTACTICALLY.IS_INTEGER ( TOKEN ) then
      RETURN_TYPE :=
      ( TYPE_IS      => RESULT.IS_UNKNOWN,
        LOCATION     => RESULT.IN_PROGRAM,
        UNKNOWN_TYPE => ( CLASS => DDL_DEFINITIONS.INT_EGER ) );
    else
      RETURN_TYPE :=
      ( TYPE_IS      => RESULT.IS_UNKNOWN,
        LOCATION     => RESULT.IN_PROGRAM,
        UNKNOWN_TYPE => ( CLASS => DDL_DEFINITIONS.FL_OAT ) );
    end if;
  end PROCESS_PRIMARY_NUMERIC_LITERAL;

  procedure PROCESS_PRIMARY_STRING_LITERAL
            ( TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  begin
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
```

```
  TENTATIVE_FUNCTIONS := TENTATIVE.FUNCTION_LIST_CREATOR;
  RETURN_TYPE :=
  ( TYPE_IS      => RESULT.IS_UNKNOWN,
    LOCATION     => RESULT.IN_PROGRAM,
    UNKNOWN_TYPE => ( CLASS => DDL_DEFINITIONS.STR_ING ) );
end PROCESS_PRIMARY_STRING_LITERAL;

procedure PROCESS_PRIMARY
          ( FROM                : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY     : in  BOOLEAN;
            LOCATION            : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE  : out BOOLEAN ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  case TOKEN.KIND is
    when LEXICAL_ANALYZER.IDENTIFIER =>
      PROCESS_PRIMARY_IDENTIFIER
      ( TOKEN , FROM , THIS_SCOPE_ONLY, LOCATION , TENTATIVE_FUNCTIONS ,
        RETURN_TYPE , SAW_DATABASE_VALUE );
    when LEXICAL_ANALYZER.NUMERIC_LITERAL =>
      PROCESS_PRIMARY_NUMERIC_LITERAL
      ( TOKEN , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when LEXICAL_ANALYZER.CHARACTER_LITERAL =>
      PROCESS_PRIMARY_CHARACTER_LITERAL
      ( FROM , TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when LEXICAL_ANALYZER.STRING_LITERAL =>
      PROCESS_PRIMARY_STRING_LITERAL ( TENTATIVE_FUNCTIONS , RETURN_TYPE );
      SAW_DATABASE_VALUE := FALSE;
    when LEXICAL_ANALYZER.DELIMITER =>
      PROCESS_PRIMARY_DELIMITER
      ( TOKEN , FROM , THIS_SCOPE_ONLY , LOCATION ,
        TENTATIVE_FUNCTIONS , RETURN_TYPE , SAW_DATABASE_VALUE );
    when LEXICAL_ANALYZER.RESERVED_WORD | LEXICAL_ANALYZER.END_OF_FILE =>
      REPORT_PRIMARY_ERROR ( TOKEN );
  end case;
end PROCESS_PRIMARY;

function IS_ADDING_OPERATOR ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN )
 return BOOLEAN is
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER then
    case TOKEN.DELIMITER is
      when LEXICAL_ANALYZER.PLUS | LEXICAL_ANALYZER.HYPHEN =>
        return TRUE;
```

```
      when others =>
        null;
    end case;
  end if;
  return FALSE;
end IS_ADDING_OPERATOR;

function IS_MULTIPLYING_OPERATOR ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN )
 return BOOLEAN is
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER then
    case TOKEN.DELIMITER is
      when LEXICAL_ANALYZER.STAR | LEXICAL_ANALYZER.SLASH =>
        return TRUE;
      when others =>
        null;
    end case;
  end if;
  return FALSE;
end IS_MULTIPLYING_OPERATOR;

procedure COMBINE_ARITHMETIC_OPERANDS
          ( TOKEN : in     LEXICAL_ANALYZER.LEXICAL_TOKEN;
            T1    : in out TENTATIVE.FUNCTION_LIST;
            R1    : in out RESULT.DESCRIPTOR;
            T2    : in     TENTATIVE.FUNCTION_LIST;
            R2    : in     RESULT.DESCRIPTOR ) is
  R3         : RESULT.DESCRIPTOR;
  COMPARABLE : RESULT.COMPARABILITY;
begin
  SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
  ( TOKEN , R1 , R2 , R3 , COMPARABLE );
  if COMPARABLE = RESULT.IS_COMPARABLE then
    VALIDATE_NUMERIC_PARAMETER ( TOKEN , R3 );
  end if;
  if R3.TYPE_IS = RESULT.IS_KNOWN then
    TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED ( T1 , R3.KNOWN_TYPE );
    TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED ( T2 , R3.KNOWN_TYPE );
    T1 := TENTATIVE.FUNCTION_LIST_CREATOR;
  else
    T1 := TENTATIVE.FUNCTION_LIST_MERGE ( T1 , T2 );
  end if;
  if R3.LOCATION = RESULT.IN_DATABASE then
    TENTATIVE.FUNCTION_REQUIRED_FOR_BINARY_OPERATION
    ( T1, R3, SEMANTICALLY.BINARY_SQL_OPERATION (TOKEN.DELIMITER), R1, R2 );
  end if;
  R1 := R3;
end COMBINE_ARITHMETIC_OPERANDS;
```

```
procedure PROCESS_TERM
          ( FROM                  : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY       : in  BOOLEAN;
            LOCATION              : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS   : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE           : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE    : out BOOLEAN ) is
  T1, T2 : TENTATIVE.FUNCTION_LIST;
  R1, R2 : RESULT.DESCRIPTOR;
  TOKEN  : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  LEFT_DATABASE_VALUE,
  RIGHT_DATABASE_VALUE : BOOLEAN;
begin
  PROCESS_PRIMARY
  ( FROM , THIS_SCOPE_ONLY , LOCATION , T1 , R1 , LEFT_DATABASE_VALUE );
  loop
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  exit when not IS_MULTIPLYING_OPERATOR ( TOKEN );
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    PROCESS_PRIMARY
    ( FROM , THIS_SCOPE_ONLY , LOCATION , T2 , R2 , RIGHT_DATABASE_VALUE );
    LEFT_DATABASE_VALUE := LEFT_DATABASE_VALUE or RIGHT_DATABASE_VALUE;
    COMBINE_ARITHMETIC_OPERANDS ( TOKEN , T1 , R1 , T2 , R2 );
  end loop;
  TENTATIVE_FUNCTIONS := T1;
  RETURN_TYPE := R1;
  SAW_DATABASE_VALUE := LEFT_DATABASE_VALUE;
end PROCESS_TERM;

procedure PROCESS_PLUS_OR_MINUS_TERM
          ( TOKEN                 : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM                  : in  FROM_CLAUSE.INFORMATION;
            THIS_SCOPE_ONLY       : in  BOOLEAN;
            LOCATION              : in  SEMANTICALLY.LOCATION_RESTRICTION;
            TENTATIVE_FUNCTIONS   : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE           : out RESULT.DESCRIPTOR;
            SAW_DATABASE_VALUE    : out BOOLEAN ) is
  T1     : TENTATIVE.FUNCTION_LIST;
  R1     : RESULT.DESCRIPTOR;
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  PROCESS_TERM
  ( FROM , THIS_SCOPE_ONLY , LOCATION , T1 , R1 , SAW_DATABASE_VALUE );
  VALIDATE_NUMERIC_PARAMETER ( TOKEN , R1 );
  if R1.LOCATION = RESULT.IN_DATABASE then
    TENTATIVE.FUNCTION_REQUIRED_FOR_UNARY_OPERATION
    ( T1 , R1 , UNARY_SQL_OPERATION ( TOKEN.DELIMITER ) , R1 );
  end if;
  TENTATIVE_FUNCTIONS := T1;
```

```
    RETURN_TYPE := R1;
  end PROCESS_PLUS_OR_MINUS_TERM;

  procedure PROCESS_ADDING_OPERATORS
            ( FROM                 : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY      : in  BOOLEAN;
              LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE          : out RESULT.DESCRIPTOR;
              SAW_DATABASE_VALUE   : out BOOLEAN ) is
    T1, T2 : TENTATIVE.FUNCTION_LIST;
    R1, R2 : RESULT.DESCRIPTOR;
    TOKEN  : LEXICAL_ANALYZER.LEXICAL_TOKEN;
    LEFT_DATABASE_VALUE,
    RIGHT_DATABASE_VALUE : BOOLEAN;
  begin
    PROCESS_TERM
    ( FROM , THIS_SCOPE_ONLY , LOCATION , T1 , R1 , LEFT_DATABASE_VALUE );
    loop
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    exit when not IS_ADDING_OPERATOR ( TOKEN );
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      PROCESS_TERM
      ( FROM , THIS_SCOPE_ONLY , LOCATION , T2 , R2 , RIGHT_DATABASE_VALUE );
      LEFT_DATABASE_VALUE := LEFT_DATABASE_VALUE or RIGHT_DATABASE_VALUE;
      COMBINE_ARITHMETIC_OPERANDS ( TOKEN , T1 , R1 , T2 , R2 );
    end loop;
    TENTATIVE_FUNCTIONS := T1;
    RETURN_TYPE := R1;
    SAW_DATABASE_VALUE := LEFT_DATABASE_VALUE;
  end PROCESS_ADDING_OPERATORS;

  procedure PROCESS_VALUE_EXPRESSION
            ( FROM                 : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY      : in  BOOLEAN;
              LOCATION             : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE          : out RESULT.DESCRIPTOR;
              SAW_DATABASE_VALUE   : out BOOLEAN ) is
    TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
     LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  begin
    if IS_ADDING_OPERATOR ( TOKEN ) then
      PROCESS_PLUS_OR_MINUS_TERM
      ( TOKEN , FROM , THIS_SCOPE_ONLY , LOCATION , TENTATIVE_FUNCTIONS ,
        RETURN_TYPE , SAW_DATABASE_VALUE );
      return;
    end if;
    PROCESS_ADDING_OPERATORS
```

```
    ( FROM , THIS_SCOPE_ONLY , LOCATION , TENTATIVE_FUNCTIONS , RETURN_TYPE ,
      SAW_DATABASE_VALUE );
  end PROCESS_VALUE_EXPRESSION;

  procedure PROCESS_VALUE_EXPRESSION
            ( FROM                : in  FROM_CLAUSE.INFORMATION;
              THIS_SCOPE_ONLY     : in  BOOLEAN;
              LOCATION            : in  SEMANTICALLY.LOCATION_RESTRICTION;
              TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
    DUMMY : BOOLEAN;
  begin
    PROCESS_VALUE_EXPRESSION
    ( FROM , THIS_SCOPE_ONLY , LOCATION , TENTATIVE_FUNCTIONS, RETURN_TYPE ,
      DUMMY );
  end PROCESS_VALUE_EXPRESSION;

end EXPRESSION;
```

3.11.74 package ddl_schema_io_errors_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS;
use  IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS;

package IO_ERRORS is

  procedure OPEN_ERROR         -- internal, exceptions for OEPN_SCHEMA_UNIT
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            MESSAGE : in STRING;
            NAME    : in STRING);

  procedure READ_ERROR         -- internal, exceptions for NEXT_LINE
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            MESSAGE : in STRING;
            NAME    : in LIBRARY_UNIT_NAME_STRING);

  procedure CLOSE_ERROR        -- internal, exceptions for CLOSE_SCHEMA_UNIT
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            MESSAGE : in STRING;
            NAME    : in LIBRARY_UNIT_NAME_STRING);

  procedure PRINT_ERROR_ERROR  -- internal, exceptions for PRINT_ERROR&TO_FILE
           (MESSAGE : in STRING);

  procedure PRINT_MESSAGE_ERROR -- internal, exceptions for PRINT_MESSAGE
           (MESSAGE : in STRING);

  procedure INPUT_ERROR        -- internal, exceptions for GET_TERMINAL_INPUT
           (MESSAGE : in STRING);
```

```
  procedure OPEN_OUTPUT_FILE_ERROR -- internal, exceptions for OPEN_OUTPUT_FILE
           (MESSAGE : in STRING;
            NAME    : in STRING);

  procedure CLOSE_OUTPUT_FILE_ERROR -- internal, exceptions for
                                    -- CLOSE_OUTPUT_FILE
           (MESSAGE : in STRING);

end IO_ERRORS;
```

3.11.75 package scans.ada

```
-- scans.ada - driver for DML processing of Ada/SQL Application Scanner

package SCAN_DML is

-- This package contains the driver for scanning an application compilation
-- unit for Ada/SQL DML statements.  The procedure, APPLICATION_UNIT, is
-- called with the name of the file which contains the compilation unit to be
-- scanned and the name of the file which is to contain the resulting listing
-- (if any).  Note that if UNIT_FILENAME = "STANDARD_INPUT" the scanner will
-- seek the input for the application compilation unit from the file
-- TEXT_IO.STANDARD_INPUT and that if LISTING_FILENAME = "STANDARD_OUTPUT" the
-- scanner will produce the output listing to the file TEXT_IO.STANDARD_OUTPUT.

   procedure PROCESS_APPLICATION_UNIT
      (UNIT_FILENAME              : in STRING;
       LISTING_FILENAME           : in STRING := "";
       GENERATED_PACKAGE_FILENAME : in STRING);

end SCAN_DML;
```

3.11.76 package searchs.ada

```
-- searchs.ada - routine to process a search condition

with FROM_CLAUSE;
package SEARCH_CONDITION is

  procedure PROCESS_SEARCH_CONDITION ( FROM : FROM_CLAUSE.INFORMATION );

end SEARCH_CONDITION;
```

3.11.77 package statements.ada

```
package STATEMENT is
  procedure PROCESS_OPEN_STATEMENT;
  procedure PROCESS_DELETE_STATEMENT_SEARCHED;
  procedure PROCESS_UPDATE_STATEMENT_SEARCHED;
  procedure PROCESS_CLOSE_STATEMENT;
```

```
  procedure PROCESS_PACKAGE;
end STATEMENT;
```

3.11.78 package tblexprs.ada

```
with FROM_CLAUSE;

package TABLE_EXPRESSION is

  procedure PROCESS_FROM_CLAUSE
           (SCOPE : FROM_CLAUSE.INFORMATION);

  procedure PROCESS_REST_OF_TABLE_EXPRESSION
           (SCOPE : FROM_CLAUSE.INFORMATION);

end TABLE_EXPRESSION;
```

3.11.79 package selects.ada

```
with DDL_DEFINITIONS, RESULT;

package SELECT_STATEMENT is

  type TYPE_OF_COLUMN is (NAMED_COLUMN, NOT_NAMED_COLUMN);
  type SELECTED_ITEM_LIST_RECORD (COLUMN_TYPE : TYPE_OF_COLUMN := NAMED_COLUMN);
  type SELECTED_ITEM_LIST is access SELECTED_ITEM_LIST_RECORD;
  type SELECTED_ITEM_LIST_RECORD
         (COLUMN_TYPE : TYPE_OF_COLUMN := NAMED_COLUMN) is
    record
      NEXT_ITEM : SELECTED_ITEM_LIST;
      RESULT_DESCRIPTOR : RESULT.DESCRIPTOR;
      STRONGLY_TYPED_DES : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      case COLUMN_TYPE is
        when NAMED_COLUMN =>
          COLUMN_NAME : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
        when NOT_NAMED_COLUMN =>
          null;
      end case;
    end record;

  type LIST_OF_COLUMNS_RECORD;
  type LIST_OF_COLUMNS is access LIST_OF_COLUMNS_RECORD;
  type LIST_OF_COLUMNS_RECORD is
    record
      COLUMN_DES  : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      NEXT_COLUMN : LIST_OF_COLUMNS;
    end record;

  procedure PROCESS_SELECT_STATEMENT;
```

```
  procedure PROCESS_DECLARE_CURSOR;

  procedure PROCESS_INSERT_INTO;

  procedure PROCESS_FETCH;

end SELECT_STATEMENT;
```

3.11.80 package selectb.ada

```
with ADA_SQL_FUNCTION_DEFINITIONS, CORRELATION, DDL_DEFINITIONS, EXPRESSION,
     FROM_CLAUSE, GENERATED_FUNCTIONS, INDEX_SUBTYPE, INTO, LEXICAL_ANALYZER,
     NAME, PREDEFINED, PREDEFINED_TYPE, QUALIFIED_NAME, RESULT, SELEC,
     SEMANTICALLY, SYNTACTICALLY, TABLE, TABLE_EXPRESSION, TENTATIVE,
     UNQUALIFIED_NAME, TEXT_IO;
use  DDL_DEFINITIONS, GENERATED_FUNCTIONS, LEXICAL_ANALYZER, NAME, RESULT,
     TEXT_IO;

package body SELECT_STATEMENT is

-------------------------------------------------------------------------
--  PROCESS_RESULT_SPECIFICATION     result_specification
--                                       result_program_variable
--                                       [ , last_variable ]

  procedure PROCESS_RESULT_SPECIFICATION
           (FROM_INFO    : FROM_CLAUSE.INFORMATION;
            RESULT_TYPE  : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR) is

    NAME_INFO : NAME.INFORMATION;
    LAST_INFO : NAME.INFORMATION;
    TOKEN     : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    --**PUT_LINE ("*****enter PROCESS_RESULT_SPECIFICATION");
    NAME_INFO := NAME.AT_CURRENT_INPUT_POINT (FROM_INFO,
                 NAME.IS_PROGRAM_VARIABLE, TRUE, FALSE);
    SYNTACTICALLY.GOBBLE_NAME (NAME_INFO);
    INTO.REQUIRED_FOR (NAME_INFO.VARIABLE_TYPE.TYPE_IS.BASE_TYPE.FULL_NAME);
    if NAME_INFO.VARIABLE_TYPE.TYPE_IS.TY_PE = DDL_DEFINITIONS.STR_ING then
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        LAST_INFO := NAME.AT_CURRENT_INPUT_POINT (FROM_INFO,
                     NAME.IS_PROGRAM_VARIABLE, TRUE, FALSE);
        SYNTACTICALLY.GOBBLE_NAME (LAST_INFO);
        --INDEX_SUBTYPE.REQUIRED_FOR (NAME_INFO.VARIABLE_TYPE.TYPE_IS.BASE_TYPE);
        if NAME_INFO.VARIABLE_TYPE.TYPE_IS.INDEX_TYPE.BASE_TYPE /=
           LAST_INFO.VARIABLE_TYPE.TYPE_IS.BASE_TYPE then
```

```
          --**PUT_LINE ("*****exit PROCESS_RESULT_SPECIFICATION");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Last_variable must be of the same type as the index of the string");
        end if;
      else
        --**PUT_LINE ("*****exit PROCESS_RESULT_SPECIFICATION");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Illegal index type for result program variable");
      end if;
    end if;
    RESULT_TYPE := NAME_INFO.VARIABLE_TYPE.TYPE_IS.BASE_TYPE;
    --**PUT_LINE ("*****exit PROCESS_RESULT_SPECIFICATION");
  end PROCESS_RESULT_SPECIFICATION;

------------------------------------------------------------------------
-- PROCESS_INTO_STATEMENTS        INTO (result_specification) ;

  procedure PROCESS_INTO_STATEMENTS
          (ITEM_LIST : SELECTED_ITEM_LIST;
           FROM_INFO : FROM_CLAUSE.INFORMATION) is

    TOKEN        : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    CURRENT_ITEM : SELECTED_ITEM_LIST := ITEM_LIST;
    RESULT_TYPE  : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
    COMPARE_DES  : RESULT.DESCRIPTOR;
    CAN_COMPARE  : RESULT.COMPARABILITY;

  begin
    --**PUT_LINE ("*****enter PROCESS_INTO_STATEMENTS");
    loop
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
        TOKEN.ID.all = "INTO" then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      elsif CURRENT_ITEM /= null then
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "More objects selected then there are INTO statements");
      else
        exit;
      end if;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
        TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting left parenthesis");
```

```
      end if;
      PROCESS_RESULT_SPECIFICATION (FROM_INFO, RESULT_TYPE);
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
        TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting right parenthesis");
      end if;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
        TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting semicolon");
      end if;
      if CURRENT_ITEM = null then
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "More INTO statements then objects selected");
      end if;
      RESULT.COMBINED_TYPE (RESULT_TYPE.BASE_TYPE,
             CURRENT_ITEM.RESULT_DESCRIPTOR, COMPARE_DES, CAN_COMPARE);
      if CAN_COMPARE = RESULT.IS_COMPARABLE then
        CURRENT_ITEM := CURRENT_ITEM.NEXT_ITEM;
      else
        --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Variable type of INTO argument not compatible with " &
        "type of object selected in select_item");
      end if;
    end loop;
    --**PUT_LINE ("*****exit PROCESS_INTO_STATEMENTS");
  end PROCESS_INTO_STATEMENTS;

-------------------------------------------------------------------
-- POSSIBLE_COLUMN

  function POSSIBLE_COLUMN
          (FROM_INFO : FROM_CLAUSE.INFORMATION)
           return DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR is

    FULL_NAME     : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;
    NAME_INFO     : NAME.INFORMATION;
    TOKEN         : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
```

```
begin
  --**PUT_LINE ("*****enter POSSIBLE_COLUMN");
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
    NAME_INFO := NAME.AT_CURRENT_INPUT_POINT (FROM_INFO,
                 NAME.IS_COLUMN_SPECIFICATION, TRUE, FALSE, FALSE);
    if NAME_INFO.KIND = NAME.OF_QUALIFIED_COLUMN then
      --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
      return NAME_INFO.QUALIFIED_COLUMN;
    elsif NAME_INFO.KIND = NAME.OF_CORRELATED_COLUMN then
      --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
      return NAME_INFO.CORRELATED_COLUMN;
    elsif NAME_INFO.KIND = NAME.OF_UNQUALIFIED_COLUMN then
      --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
      return NAME_INFO.UNQUALIFIED_COLUMN;
    else
      --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
      return null;
    end if;
    --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
  else
    return null;
  end if;
exception
  when LEXICAL_ANALYZER.SYNTAX_ERROR =>
            --**PUT_LINE ("*****exit POSSIBLE_COLUMN");
            return null;
end POSSIBLE_COLUMN;

--------------------------------------------------------------------------
-- PROCESS_SELECT_LIST

  procedure PROCESS_SELECT_LIST
          (FROM_INFO      : FROM_CLAUSE.INFORMATION;
           SELECTED_ITEMS : out SELECTED_ITEM_LIST) is

    TENTATIVE_FUNCTION_LIST : TENTATIVE.FUNCTION_LIST;
    RESULT_DES              : RESULT.DESCRIPTOR;
    FIRST_ITEM              : SELECTED_ITEM_LIST := null;
    NEW_ITEM                : SELECTED_ITEM_LIST := null;
    LAST_ITEM               : SELECTED_ITEM_LIST := null;
    POSS_COLUMN             : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR
                              := null;
    COL_TYPE                : TYPE_OF_COLUMN;
    TOKEN                   : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    COLUMN_COUNT            : NATURAL := 0;
    LEFT_OPERAND_KIND,
    RIGHT_OPERAND_KIND      : GENERATED_FUNCTIONS.OPERAND_KIND;
    LEFT_OPERAND_TYPE,
```

```
  RIGHT_OPERAND_TYPE       : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
  procedure SET_PARAMETER
            ( SELECTED_ITEM : in  SELECTED_ITEM_LIST;
              OPERAND_KIND  : out GENERATED_FUNCTIONS.OPERAND_KIND;
              OPERAND_TYPE  : out
               DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR ) is
  begin
    if SELECTED_ITEM.RESULT_DESCRIPTOR.LOCATION = RESULT.IN_PROGRAM then
      OPERAND_KIND := GENERATED_FUNCTIONS.O_USER_TYPE;
      OPERAND_TYPE := SELECTED_ITEM.STRONGLY_TYPED_DES.FULL_NAME;
    else
      OPERAND_KIND := GENERATED_FUNCTIONS.O_SQL_OBJECT;
      OPERAND_TYPE := null;
    end if;
  end SET_PARAMETER;

begin
  --**PUT_LINE ("*****enter SELECT_LIST");
  loop
    POSS_COLUMN := POSSIBLE_COLUMN (FROM_INFO);
    EXPRESSION.PROCESS_VALUE_EXPRESSION (FROM_INFO, TRUE,
          SEMANTICALLY.ANY_VALUE, TENTATIVE_FUNCTION_LIST,
          RESULT_DES);
    COLUMN_COUNT := COLUMN_COUNT + 1;
    if POSS_COLUMN /= null and then
       RESULT_DES.LOCATION = RESULT.IN_DATABASE then
      COL_TYPE := NAMED_COLUMN;
      NEW_ITEM := new SELECTED_ITEM_LIST_RECORD '
                     (COLUMN_TYPE        => NAMED_COLUMN,
                      NEXT_ITEM          => null,
                      RESULT_DESCRIPTOR  => RESULT_DES,
                      STRONGLY_TYPED_DES => SEMANTICALLY.STRONGLY_TYPE
                                           (RESULT_DES),
                      COLUMN_NAME        => POSS_COLUMN);
    else
      COL_TYPE := NOT_NAMED_COLUMN;
      NEW_ITEM := new SELECTED_ITEM_LIST_RECORD '
                     (COLUMN_TYPE        => NOT_NAMED_COLUMN,
                      NEXT_ITEM          => null,
                      RESULT_DESCRIPTOR  => RESULT_DES,
                      STRONGLY_TYPED_DES => SEMANTICALLY.STRONGLY_TYPE
                                           (RESULT_DES));
    end if;
    if FIRST_ITEM = null then
      FIRST_ITEM := NEW_ITEM;
      LAST_ITEM := NEW_ITEM;
      SELECTED_ITEMS := FIRST_ITEM;
    else
      LAST_ITEM.NEXT_ITEM := NEW_ITEM;
```

```
          LAST_ITEM := NEW_ITEM;
        end if;
        TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT (TENTATIVE_FUNCTION_LIST);
        if COLUMN_COUNT = 1 then
          null;
        elsif COLUMN_COUNT = 2 then
          SET_PARAMETER (FIRST_ITEM, LEFT_OPERAND_KIND, LEFT_OPERAND_TYPE);
          SET_PARAMETER (LAST_ITEM, RIGHT_OPERAND_KIND, RIGHT_OPERAND_TYPE);
          GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
              (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND, LEFT_OPERAND_KIND,
               LEFT_OPERAND_TYPE, RIGHT_OPERAND_KIND, RIGHT_OPERAND_TYPE,
               GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
        else
          SET_PARAMETER (LAST_ITEM, RIGHT_OPERAND_KIND, RIGHT_OPERAND_TYPE);
          GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
              (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND,
               GENERATED_FUNCTIONS.O_SQL_OBJECT, null, RIGHT_OPERAND_KIND,
               RIGHT_OPERAND_TYPE, GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
        end if;
        TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
          TOKEN.DELIMITER = LEXICAL_ANALYZER.AMPERSAND then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        elsif TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
              TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
          exit;
        else
          exit;
        end if;
      end loop;
      --**PUT_LINE ("*****exit SELECT_LIST");
    end PROCESS_SELECT_LIST;

  ---------------------------------------------------------------------------
  -- PROCESS_SELECT_LIST_OR_STAR

    procedure PROCESS_SELECT_LIST_OR_STAR
             (FROM_INFO              : FROM_CLAUSE.INFORMATION;
              SELECTED_ITEMS         : out SELECTED_ITEM_LIST;
              SELECT_STAR            : out BOOLEAN) is

      FIRST_ITEM               : SELECTED_ITEM_LIST := null;
      NEW_ITEM                 : SELECTED_ITEM_LIST := null;
      LAST_ITEM                : SELECTED_ITEM_LIST := null;
      COL_TYPE                 : TYPE_OF_COLUMN := NAMED_COLUMN;
      TOKEN                    : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
      TABLE_IS                 : FROM_CLAUSE.TABLE_ENTRY;
      MORE_TABLES              : BOOLEAN := TRUE;
```

```
    TABLE_DES                : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    COLUMN_DES               : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    COLUMN_FULL_NAME         : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    RESULT_DES               : RESULT.DESCRIPTOR (RESULT.IS_KNOWN);

  begin
    --**PUT_LINE ("*****enter PROCESS_SELECT_LIST_OR_STAR");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.CHARACTER_LITERAL and then
       TOKEN.CHARACTER_VALUE = '*' then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      SELECT_STAR := TRUE;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      end if;
    else
      SELECT_STAR := FALSE;
      PROCESS_SELECT_LIST (FROM_INFO, SELECTED_ITEMS);
      --**PUT_LINE ("*****exit PROCESS_SELECT_LIST_OR_STAR");
      return;
    end if;
    TABLE_IS := FROM_CLAUSE.TABLES_AT_CURRENT_SCOPE (FROM_INFO);
    while MORE_TABLES loop
      FROM_CLAUSE.NEXT_TABLE (TABLE_IS, MORE_TABLES, TABLE_DES);
      COLUMN_DES := TABLE_DES.FIRST_COMPONENT;
      while COLUMN_DES /= null loop
        COLUMN_FULL_NAME := COLUMN_DES.FULL_NAME;
        RESULT_DES.LOCATION := RESULT.IN_DATABASE;
        RESULT_DES.KNOWN_TYPE := COLUMN_DES.BASE_TYPE;
        NEW_ITEM := new SELECTED_ITEM_LIST_RECORD '
                       (COLUMN_TYPE         => NAMED_COLUMN,
                        NEXT_ITEM           => null,
                        RESULT_DESCRIPTOR   => RESULT_DES,
                        STRONGLY_TYPED_DES  => SEMANTICALLY.STRONGLY_TYPE
                                              (RESULT_DES),
                        COLUMN_NAME         => COLUMN_FULL_NAME);
        if FIRST_ITEM = null then
          FIRST_ITEM := NEW_ITEM;
          LAST_ITEM := NEW_ITEM;
          SELECTED_ITEMS := FIRST_ITEM;
        else
          LAST_ITEM.NEXT_ITEM := NEW_ITEM;
          LAST_ITEM := NEW_ITEM;
        end if;
        COLUMN_DES := COLUMN_DES.NEXT_ONE;
      end loop;
    end loop;
```

```
    --**PUT_LINE ("*****exit PROCESS_SELECT_LIST_OR_STAR");
  end PROCESS_SELECT_LIST_OR_STAR;

------------------------------------------------------------------------------
-- CHECK_STRONGLY_TYPED_FOR_ITEM_LIST

  procedure CHECK_STRONGLY_TYPED_FOR_ITEM_LIST
           (SELECTED_ITEMS : SELECTED_ITEM_LIST;
            TOKEN          : LEXICAL_ANALYZER.LEXICAL_TOKEN) is

    ITEM : SELECTED_ITEM_LIST := SELECTED_ITEMS;

  begin
    --**PUT_LINE ("*****enter CHECK_STRONGLY_TYPED_FOR_ITEM_LIST");
    while ITEM /= null loop
      if ITEM.COLUMN_TYPE = NAMED_COLUMN and then
         ITEM.STRONGLY_TYPED_DES = null then
        ITEM.STRONGLY_TYPED_DES := ITEM.COLUMN_NAME.TYPE_IS.BASE_TYPE;
      elsif ITEM.STRONGLY_TYPED_DES = null then
        if ITEM.RESULT_DESCRIPTOR.TYPE_IS = IS_KNOWN then
          ITEM.STRONGLY_TYPED_DES := ITEM.RESULT_DESCRIPTOR.KNOWN_TYPE;
        else
          case ITEM.RESULT_DESCRIPTOR.UNKNOWN_TYPE.CLASS is
            when DDL_DEFINITIONS.INT_EGER    =>
                 ITEM.STRONGLY_TYPED_DES := PREDEFINED_TYPE.STANDARD.INTEGER;
            when DDL_DEFINITIONS.FL_OAT      =>
                 ITEM.STRONGLY_TYPED_DES := PREDEFINED_TYPE.STANDARD.FLOAT;
            when DDL_DEFINITIONS.STR_ING     =>
                 ITEM.STRONGLY_TYPED_DES := PREDEFINED_TYPE.STANDARD.STRING;
            when DDL_DEFINITIONS.ENUMERATION =>
                 ITEM.STRONGLY_TYPED_DES := PREDEFINED_TYPE.STANDARD.CHARACTER;
                 LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
                 "A declare cursor or select statement cannot select an " &
                 "enumeration value that is not unique");
          end case;
        end if;
      end if;
      ITEM := ITEM.NEXT_ITEM;
    end loop;
    --**PUT_LINE ("*****exit CHECK_STRONGLY_TYPED_FOR_ITEM_LIST");
  end CHECK_STRONGLY_TYPED_FOR_ITEM_LIST;

---------------------------------------------------------------------------
-- PROCESS_SELECT_STATEMENT      SELEC
--                               SELECT_ALL
--                               SELECT_DISTINCT
--                               ( select_list , table_expression ) ;
--                               INTO ( result_specification ) ;
--                               INTO ...
```

```
--
--                                           table_expression
--                                              from_clause
--                                              where_clause
--                                              group_by_clause
--                                              having_clause

  procedure PROCESS_SELECT_STATEMENT is

    TOKEN            : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    SELECT_TYPE      : SELEC.ROUTINE_NAME;
    FROM_INFO        : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
    SELECTED_ITEMS   : SELECTED_ITEM_LIST;
    SELECT_STAR      : BOOLEAN;

  begin
    --**PUT_LINE ("*****enter PROCESS_SELECT_STATEMENT");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
      if TOKEN.ID.all = "SELEC" then
        SELECT_TYPE := SELEC.SELEC;
      elsif TOKEN.ID.all = "SELECT_ALL" then
        SELECT_TYPE := SELEC.SELECT_ALL;
      elsif TOKEN.ID.all = "SELECT_DISTINCT" then
        SELECT_TYPE := SELEC.SELECT_DISTINCT;
      else
        --**PUT_LINE ("*****exit PROCESS_SELECT_STATEMENT");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
      end if;
    else
      --**PUT_LINE ("*****exit PROCESS_SELECT_STATEMENT");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
    end if;
    SYNTACTICALLY.SKIP_SELECT_CLAUSE;
    TABLE_EXPRESSION.PROCESS_FROM_CLAUSE (FROM_INFO);
    LEXICAL_ANALYZER.RESTORE_SKIPPED_TOKENS;
    PROCESS_SELECT_LIST_OR_STAR (FROM_INFO, SELECTED_ITEMS, SELECT_STAR);
    TABLE_EXPRESSION.PROCESS_REST_OF_TABLE_EXPRESSION (FROM_INFO);
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
```

```
          --**PUT_LINE ("*****exit PROCESS_SELECT_STATEMENT");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Expecting semicolon");
        end if;
      else
        --**PUT_LINE ("*****exit PROCESS_SELECT_STATEMENT");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting right parenthesis");
      end if;
      PROCESS_INTO_STATEMENTS (SELECTED_ITEMS, FROM_INFO);
      CHECK_STRONGLY_TYPED_FOR_ITEM_LIST (SELECTED_ITEMS, TOKEN);
      if SELECT_STAR then
        SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.STAR, SELEC.PROCEDURE_CALL, null);
      elsif SELECTED_ITEMS.NEXT_ITEM = null and then
            SELECTED_ITEMS.RESULT_DESCRIPTOR.LOCATION = IN_PROGRAM then
        if SELECTED_ITEMS.RESULT_DESCRIPTOR.TYPE_IS = IS_KNOWN then
          SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE,
                SELEC.PROCEDURE_CALL,
                SELECTED_ITEMS.RESULT_DESCRIPTOR.KNOWN_TYPE.FULL_NAME);
        else
          SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE,
                              SELEC.PROCEDURE_CALL,
                              SELECTED_ITEMS.STRONGLY_TYPED_DES.FULL_NAME);
        end if;
      else
        SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.SQL_OBJECT, SELEC.PROCEDURE_CALL,
                            null);
      end if;
      --**PUT_LINE ("*****exit PROCESS_SELECT_STATEMENT");
    end PROCESS_SELECT_STATEMENT;


--------------------------------------------------------------------------
-- COLUMN_IN_SELECTED_LIST (SELECTED_ITEMS, COLUMN_DES)

    function COLUMN_IN_SELECTED_LIST
            (SELECTED_ITEMS : SELECTED_ITEM_LIST;
             COLUMN_DES     : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR)
             return           BOOLEAN is

      ITEMS : SELECTED_ITEM_LIST := SELECTED_ITEMS;

    begin
      --**PUT_LINE ("** **enter COLUMN_IN_SELECTED_LIST");
      while ITEMS /= null loop
        if ITEMS.COLUMN_TYPE = NAMED_COLUMN and then
           ITEMS.COLUMN_NAME = COLUMN_DES then
          --**PUT_LINE ("*****exit COLUMN_IN_SELECTED_LIST");
          return TRUE;
```

```
      end if;
      ITEMS := ITEMS.NEXT_ITEM;
    end loop;
    --**PUT_LINE ("*****exit COLUMN_IN_SELECTED_LIST");
    return FALSE;
  end COLUMN_IN_SELECTED_LIST;

------------------------------------------------------------------------
-- COUNT_SELECTED_ITEMS

  function COUNT_SELECTED_ITEMS
          (SELECTED_ITEMS : SELECTED_ITEM_LIST)
           return           NATURAL is

    ITEM : SELECTED_ITEM_LIST := SELECTED_ITEMS;
    COUNT : NATURAL := 0;

  begin
    --**PUT_LINE ("*****enter COUNT_SELECTED_ITEMS");
    while ITEM /= null loop
      COUNT := COUNT + 1;
      ITEM := ITEM.NEXT_ITEM;
    end loop;
    --**PUT_LINE ("*****exit COUNT_SELECTED_ITEMS");
    return COUNT;
  end COUNT_SELECTED_ITEMS;

----------------------------------------------------------------------------
--  PROCESS_SORT_COLUMN_SPECIFICATION

  procedure PROCESS_SORT_COLUMN_SPECIFICATION
           (FROM_INFO           : FROM_CLAUSE.INFORMATION;
            SELECTED_ITEMS      : SELECTED_ITEM_LIST;
            RESULT_TYPE         : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            SELECTED_ITEM_COUNT : NATURAL) is

    TOKEN             : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    NAME_INFO         : NAME.INFORMATION;
    COLUMN_ALONE      : BOOLEAN := FALSE;
    COLUMN_TABLE      : BOOLEAN := FALSE;
    COLUMN_DES        : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR := null;

  begin
    --**PUT_LINE ("*****enter PROCESS_SORT_COLUMN_SPECIFICATION");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.NUMERIC_LITERAL then
      if INTEGER'VALUE(TOKEN.IMAGE.all) > 0 and then
         INTEGER'VALUE(TOKEN.IMAGE.all) <= SELECTED_ITEM_COUNT then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
```

```
        else
          LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
          "Column number not within range of selected items");
        end if;
        RESULT_TYPE := null;
      elsif TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
        NAME_INFO := NAME.AT_CURRENT_INPUT_POINT (FROM_INFO,
                       IS_COLUMN_SPECIFICATION, TRUE, FALSE, FALSE);
        --SYNTACTICALLY.GOBBLE_NAME (NAME_INFO);
        if NAME_INFO.KIND = NAME.OF_QUALIFIED_COLUMN then
          COLUMN_TABLE := TRUE;
          COLUMN_DES := NAME_INFO.QUALIFIED_COLUMN;
          QUALIFIED_NAME.RETURNS_SQL_OBJECT (COLUMN_DES);
        elsif NAME_INFO.KIND = NAME.OF_CORRELATED_COLUMN then
          COLUMN_DES := NAME_INFO.CORRELATED_COLUMN;
          CORRELATION.COLUMN_RETURNS_SQL_OBJECT (NAME_INFO.CORRELATION_NAME,
                   COLUMN_DES);
        elsif NAME_INFO.KIND = NAME.OF_UNQUALIFIED_COLUMN then
          COLUMN_ALONE := TRUE;
          COLUMN_DES := NAME_INFO.UNQUALIFIED_COLUMN;
          UNQUALIFIED_NAME.RETURNS_SQL_OBJECT (COLUMN_DES.NAME);
        end if;
        if COLUMN_IN_SELECTED_LIST (SELECTED_ITEMS, COLUMN_DES) then
          null;
        else
          LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
          "Column is not amoung those selected");
        end if;
        SYNTACTICALLY.GOBBLE_NAME (NAME_INFO);
        RESULT_TYPE := COLUMN_DES.TYPE_IS.BASE_TYPE;
      else
        --**PUT_LINE ("*****exit PROCESS_SORT_COLUMN_SPECIFICATION");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting column number or column name");
      end if;
      --**PUT_LINE ("*****exit PROCESS_SORT_COLUMN_SPECIFICATION");
    exception
      when LEXICAL_ANALYZER.SYNTAX_ERROR =>
                  --**PUT_LINE ("*****exit PROCESS_SORT_COLUMN_SPECIFICATION")
                        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                        "Expecting column number or column name");
    end PROCESS_SORT_COLUMN_SPECIFICATION;

  ------------------------------------------------------------------------
  -- PROCESS_SORT_SPECIFICATION              sort_column_specification
  --                                   ASC  ( sort_column_specification )
  --                                   DESC ( sort_column_specification )

    procedure PROCESS_SORT_SPECIFICATION
```

```
        ( FROM_INFO           : FROM_CL  SE.INFORMATION;
          SELECTED_ITEMS      : SELECTED_ITEM_LIST;
          SELECTED_ITEM_COUNT : NATURAL;
          RESULT_TYPE         : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
          ASCENDING           : out BOOLEAN;
          DESCENDING          : out BOOLEAN) is

    RES_TYPE             : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    TOKEN                : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    DB_COLUMN_TYPE       : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                           PREDEFINED_TYPE.DATABASE.COLUMN_NUMBER;
    DB_COLUMN            : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                           DB_COLUMN_TYPE.FULL_NAME;
    ASC_DESC             : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
    IS_ASC_OR_DESC       : BOOLEAN := FALSE;

  begin
    --**PUT_LINE ("*****enter PROCESS_SORT_SPECIFICATION");
    ASCENDING := FALSE;
    DESCENDING := FALSE;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
      if TOKEN.ID.all = "ASC" then
        ASCENDING := TRUE;
        IS_ASC_OR_DESC := TRUE;
        ASC_DESC := ADA_SQL_FUNCTION_DEFINITIONS.O_ASC;
      elsif TOKEN.ID.all = "DESC" then
        DESCENDING := TRUE;
        IS_ASC_OR_DESC := TRUE;
        ASC_DESC := ADA_SQL_FUNCTION_DEFINITIONS.O_DESC;
      end if;
    end if;
    if IS_ASC_OR_DESC then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
        TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_SORT_SPECIFICATION");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting left parenthesis");
      end if;
    end if;
    PROCESS_SORT_COLUMN_SPECIFICATION (FROM_INFO, SELECTED_ITEMS, RES_TYPE,
            SELECTED_ITEM_COUNT);
    RESULT_TYPE := RES_TYPE;
    if IS_ASC_OR_DESC then
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
```

```
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_SORT_SPECIFICATION");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting right parenthesis");
      end if;
    end if;
    if IS_ASC_OR_DESC then
      if RES_TYPE /= null then
        GENERATED_FUNCTIONS.ADD_UNARY_FUNCTION
            (ASC_DESC, GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
             GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
      else
        GENERATED_FUNCTIONS.ADD_UNARY_FUNCTION
            (ASC_DESC, GENERATED_FUNCTIONS.O_USER_TYPE, DB_COLUMN,
             GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
      end if;
    end if;
    --**PUT_LINE ("*****exit PROCESS_SORT_SPECIFICATION");
  end PROCESS_SORT_SPECIFICATION;

--------------------------------------------------------------------------
-- PROCESS_ORDER_BY_CLAUSE       ORDER_BY => sort_specification
--                                        [ & sort_specification ]

  procedure PROCESS_ORDER_BY_CLAUSE
           (FROM_INFO       : FROM_CLAUSE.INFORMATION;
            SELECTED_ITEMS  : SELECTED_ITEM_LIST;
            DECLARE_SQL_OBJ : out BOOLEAN) is

    RESULT_TYPE         : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    TOKEN               : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    ASCENDING           : BOOLEAN := FALSE;
    DESCENDING          : BOOLEAN := FALSE;
    SELECTED_ITEM_COUNT : NATURAL := COUNT_SELECTED_ITEMS (SELECTED_ITEMS);
    SORT_SPEC_COUNT     : NATURAL := 0;
    LEFT_PARM           : GENERATED_FUNCTIONS.OPERAND_KIND;
    RIGHT_PARM          : GENERATED_FUNCTIONS.OPERAND_KIND;
    THIS_PARM           : GENERATED_FUNCTIONS.OPERAND_KIND;
    LEFT_DES            : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    RIGHT_DES           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    THIS_DES            : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    DB_COLUMN_TYPE      : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                          PREDEFINED_TYPE.DATABASE.COLUMN_NUMBER;
    DB_COLUMN           : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR :=
                          DB_COLUMN_TYPE.FULL_NAME;
```

```
begin
  --**PUT_LINE ("*****enter PROCESS_ORDER_BY_CLAUSE");
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
     TOKEN.ID.all = "ORDER_BY" then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.ARROW then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_ORDER_BY_CLAUSE");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, "Expecting =>");
    end if;
  else
    --**PUT_LINE ("*****exit PROCESS_ORDER_BY_CLAUSE");
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, "Expecting ORDER_BY");
  end if;
  loop
    PROCESS_SORT_SPECIFICATION (FROM_INFO, SELECTED_ITEMS,
           SELECTED_ITEM_COUNT, RESULT_TYPE, ASCENDING, DESCENDING);
    SORT_SPEC_COUNT := SORT_SPEC_COUNT + 1;
    if RESULT_TYPE = null and then not ASCENDING and then not DESCENDING then
      THIS_PARM := GENERATED_FUNCTIONS.O_USER_TYPE;
      THIS_DES := DB_COLUMN;
    else
      THIS_PARM := GENERATED_FUNCTIONS.O_SQL_OBJECT;
      THIS_DES := null;
    end if;
    if SORT_SPEC_COUNT = 1 then
      LEFT_PARM := THIS_PARM;
      LEFT_DES := THIS_DES;
    else
      RIGHT_PARM := THIS_PARM;
      RIGHT_DES := THIS_DES;
    end if;
    if SORT_SPEC_COUNT > 1 then
      GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
               (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND,
                LEFT_PARM, LEFT_DES, RIGHT_PARM, RIGHT_DES,
                GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
      LEFT_PARM := GENERATED_FUNCTIONS.O_SQL_OBJECT;
      LEFT_DES := null;
    end if;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.AMPERSAND then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
```

```
      exit;
    end if;
  end loop;
  if SORT_SPEC_COUNT = 1 and then
     LEFT_PARM /= GENERATED_FUNCTIONS.O_SQL_OBJECT then
    DECLARE_SQL_OBJ := FALSE;
  else
    DECLARE_SQL_OBJ := TRUE;
  end if;
  --**PUT_LINE ("*****exit PROCESS_ORDER_BY_CLAUSE");
end PROCESS_ORDER_BY_CLAUSE;

------------------------------------------------------------------------
-- PROCESS_QUERY_SPECIFICATION_FOR_DECLARE

procedure PROCESS_QUERY_SPECIFICATION_FOR_DECLARE
         (RETURN_FROM_INFO : out FROM_CLAUSE.INFORMATION;
          SELECTED_ITEMS   : out SELECTED_ITEM_LIST) is

  TOKEN           : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  SELECT_TYPE     : SELEC.ROUTINE_NAME;
  FROM_INFO       : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
  ITEM            : SELECTED_ITEM_LIST;
  SELECT_STAR     : BOOLEAN;

begin
  --**PUT_LINE ("*****enter PROCESS_QUERY_SPECIFICATION_FOR_DECLARE");
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
    if TOKEN.ID.all = "SELEC" then
      SELECT_TYPE := SELEC.SELEC;
    elsif TOKEN.ID.all = "SELECT_ALL" then
      SELECT_TYPE := SELEC.SELECT_ALL;
    elsif TOKEN.ID.all = "SELECT_DISTINCT" then
      SELECT_TYPE := SELEC.SELECT_DISTINCT;
    else
      --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_DECLARE");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
    end if;
  else
    --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_DECLARE");
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
    "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
  end if;
  SYNTACTICALLY.SKIP_SELECT_CLAUSE;
  TABLE_EXPRESSION.PROCESS_FROM_CLAUSE (FROM_INFO);
  LEXICAL_ANALYZER.RESTORE_SKIPPED_TOKENS;
  PROCESS_SELECT_LIST_OR_STAR (FROM_INFO, ITEM, SELECT_STAR);
```

```
    TABLE_EXPRESSION.PROCESS_REST_OF_TABLE_EXPRESSION (FROM_INFO);
    CHECK_STRONGLY_TYPED_FOR_ITEM_LIST (ITEM, TOKEN);
    if SELECT_STAR then
      SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.STAR, SELEC.SQL_OBJECT, null);
    elsif ITEM.NEXT_ITEM = null and then
          ITEM.RESULT_DESCRIPTOR.LOCATION = IN_PROGRAM then
      if ITEM.RESULT_DESCRIPTOR.TYPE_IS = IS_KNOWN then
        SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE, SELEC.SQL_OBJECT,
                            ITEM.RESULT_DESCRIPTOR.KNOWN_TYPE.FULL_NAME);
      else
          SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE,
                              SELEC.SQL_OBJECT,
                              ITEM.STRONGLY_TYPED_DES.FULL_NAME);
      end if;
    else
      SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.SQL_OBJECT, SELEC.SQL_OBJECT,
                          null);
    end if;
    SELECTED_ITEMS := ITEM;
    RETURN_FROM_INFO := FROM_INFO;
    --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_DECLARE");
  end PROCESS_QUERY_SPECIFICATION_FOR_DECLARE;

-------------------------------------------------------------------------
-- PROCESS_QUERY_EXPRESSION

  procedure PROCESS_QUERY_EXPRESSION
           (FROM_INFO      : out FROM_CLAUSE.INFORMATION;
            SELECTED_ITEMS : out SELECTED_ITEM_LIST) is
  begin
    --**PUT_LINE ("*****enter PROCESS_QUERY_EXPRESSION");
    PROCESS_QUERY_SPECIFICATION_FOR_DECLARE (FROM_INFO, SELECTED_ITEMS);
    --**PUT_LINE ("*****exit PROCESS_QUERY_EXPRESSION");
  end PROCESS_QUERY_EXPRESSION;

-------------------------------------------------------------------------
-- PROCESS_CURSOR_SPECIFICATION     query_expression [ , order-by-clause]

  procedure PROCESS_CURSOR_SPECIFICATION
           (DECLARE_SQL_OBJ : out BOOLEAN) is

    CURSOR_INFO     : NAME.INFORMATION;
    TOKEN           : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    FROM_INFO       : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
    SELECTED_ITEMS  : SELECTED_ITEM_LIST;

  begin
    --**PUT_LINE ("*****enter PROCESS_CURSOR_SPECIFICATION");
    PROCESS_QUERY_EXPRESSION (FROM_INFO, SELECTED_ITEMS);
```

```
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
      TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        PROCESS_ORDER_BY_CLAUSE (FROM_INFO, SELECTED_ITEMS, DECLARE_SQL_OBJ);
      else
        DECLARE_SQL_OBJ := TRUE;
      end if;
    else -- not )
      --**PUT_LINE ("*****exit PROCESS_CURSOR_SPECIFICATION");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting right parenthesis");
    end if;
    --**PUT_LINE ("*****exit PROCESS_CURSOR_SPECIFICATION");
  end PROCESS_CURSOR_SPECIFICATION;

------------------------------------------------------------------------
-- PROCESS_CURSOR_NAME                          identifier

  procedure PROCESS_CURSOR_NAME
           (ISSUE_DIAGNOSTICS : BOOLEAN := TRUE) is

    CURSOR_INFO : NAME.INFORMATION;
    TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    --**PUT_LINE ("*****enter PROCESS_CURSOR_NAME");
    CURSOR_INFO := NAME.AT_CURRENT_INPUT_POINT (null,
                   NAME.IS_PROGRAM_VARIABLE, TRUE, FALSE, FALSE);
    if CURSOR_INFO.VARIABLE_TYPE.TYPE_IS.BASE_TYPE =
                   PREDEFINED_TYPE.CURSOR_DEFINITION.CURSOR_NAME then
      SYNTACTICALLY.GOBBLE_NAME (CURSOR_INFO);
    else
      --**PUT_LINE ("*****exit PROCESS_CURSOR_NAME");
      raise LEXICAL_ANALYZER.SYNTAX_ERROR;
    end if;
    --**PUT_LINE ("*****exit PROCESS_CURSOR_NAME");
  exception
    when SYNTAX_ERROR => --**PUT_LINE ("*****exit PROCESS_CURSOR_NAME");
                         if ISSUE_DIAGNOSTICS then
                           TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
                           LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                           "Expecting variable of cursor name type");
                         else
                           raise LEXICAL_ANALYZER.SYNTAX_ERROR;
```

```
                      end if;
  end PROCESS_CURSOR_NAME;

---------------------------------------------------------------------------
-- PROCESS_DECLARE_CURSOR                   DECLARE
--                                          (cursor_name,
--                                           CURSOR_FOR => cursor_specification);

  procedure PROCESS_DECLARE_CURSOR is

    TOKEN           : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    DECLARE_SQL_OBJ : BOOLEAN;

  begin
    --**PUT_LINE ("*****enter PROCESS_DECLARE_CURSOR");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
       TOKEN.ID.all = "DECLAR" then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
        TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        PROCESS_CURSOR_NAME;
        TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
          TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
          TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
          if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
             TOKEN.ID.all = "CURSOR_FOR" then
            LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
            TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
            if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
              TOKEN.DELIMITER = LEXICAL_ANALYZER.ARROW then
              LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
              PROCESS_CURSOR_SPECIFICATION (DECLARE_SQL_OBJ);
              TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
              if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
                TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
                LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
                TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
                if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
                  TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
                  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
                else -- not ;
                  --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
                  LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                  "Expecting semi colon");
```

```
                end if;
              else -- not )
                --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
                LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                "Expecting right parenthesis");
              end if;
            else -- not =>
              --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
              LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
              "Expecting =>");
            end if;
          else -- not CURSOR_FOR
            --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
            LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
            "Expecting CURSOR_FOR");
          end if;
        else -- not ,
          --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Expecting comma");
        end if;
      else -- not (
        --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting left parenthesis");
      end if;
    else -- not "DECLARE"
      --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting DECLARE");
    end if;
    if DECLARE_SQL_OBJ then
      PREDEFINED.TEXT_REQUIRED_FOR
         (PREDEFINED.DECLAR_PROCEDURE_WITH_SQL_OBJECT_ORDER_BY);
    else
      PREDEFINED.TEXT_REQUIRED_FOR
         (PREDEFINED.DECLAR_PROCEDURE_WITH_NUMERIC_ORDER_BY);
    end if;
    --**PUT_LINE ("*****exit PROCESS_DECLARE_CURSOR");
  end PROCESS_DECLARE_CURSOR;


--------------------------------------------------------------------------
-- PROCESS_FETCH                     FETCH (cursor_name);
--                                   INTO (result_specification);  ...

  procedure PROCESS_FETCH is

    TOKEN        : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
```

```
    FROM_INFO   : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
    RESULT_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

  begin
    --**PUT_LINE ("*****enter PROCESS_FETCH");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
       TOKEN.ID.all = "FETCH" then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_FETCH");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting FETCH");
    end if;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_FETCH");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting left parenthesis");
    end if;
    PROCESS_CURSOR_NAME (TRUE);
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_FETCH");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting right parenthesis");
    end if;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_FETCH");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting semicolon");
    end if;
    loop
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
         TOKEN.ID.all = "INTO" then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        exit;
```

```
        end if;
        TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
           TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        else
          --**PUT_LINE ("*****exit PROCESS_FETCH");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Expecting left parenthesis");
        end if;
        PROCESS_RESULT_SPECIFICATION (FROM_INFO, RESULT_TYPE);
        TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
           TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        else
          --**PUT_LINE ("*****exit PROCESS_FETCH");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Expecting right parenthesis");
        end if;
        TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
        if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
           TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        else
          --**PUT_LINE ("*****exit PROCESS_FETCH");
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
          "Expecting semicolon");
        end if;
      end loop;
      PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.FETCH_PROCEDURE);
      --**PUT_LINE ("*****exit PROCESS_FETCH");
    end PROCESS_FETCH;

  ----------------------------------------------------------------------
  --  COMPARE_SELECT_ITEMS_AND_COLUMN_LIST

    procedure COMPARE_SELECT_ITEMS_AND_COLUMN_LIST
             (SELECTED_ITEMS : SELECTED_ITEM_LIST;
              COLUMN_LIST    : LIST_OF_COLUMNS;
              TOKEN          : LEXICAL_ANALYZER.LEXICAL_TOKEN) is

      ITEMS       : SELECTED_ITEM_LIST := SELECTED_ITEMS;
      COLUMNS     : LIST_OF_COLUMNS := COLUMN_LIST;
      RESULT_TYPE : RESULT.DESCRIPTOR;
      CAN_COMPARE : RESULT.COMPARABILITY;

    begin
      --**PUT_LINE ("*****enter COMPARE_SELECT_ITEMS_AND_COLUMN_LIST");
```

```
    while ITEMS /= null and COLUMNS /= null loop
      RESULT.COMBINED_TYPE (COLUMNS.COLUMN_DES.BASE_TYPE,
             ITEMS.RESULT_DESCRIPTOR, RESULT_TYPE, CAN_COMPARE);
      if CAN_COMPARE = RESULT.IS_NOT_COMPARABLE then
        LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
        "Type of column not comparable to type of the selected item");
      end if;
      COLUMNS := COLUMNS.NEXT_COLUMN;
      ITEMS := ITEMS.NEXT_ITEM;
    end loop;
    if ITEMS /= null or else COLUMNS /= null then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
      "Number of selected items must equal number of columns");
    end if;
    --**PUT_LINE ("*****exit COMPARE_SELECT_ITEMS_AND_COLUMN_LIST");
  end COMPARE_SELECT_ITEMS_AND_COLUMN_LIST;

--------------------------------------------------------------------------
-- PROCESS_QUERY_SPECIFICATION_FOR_INSERT

  procedure PROCESS_QUERY_SPECIFICATION_FOR_INSERT
          (COLUMN_LIST : LIST_OF_COLUMNS) is

    TOKEN          : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    SELECT_TYPE    : SELEC.ROUTINE_NAME;
    FROM_INFO      : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
    SELECTED_ITEMS : SELECTED_ITEM_LIST;
    SELECT_STAR    : BOOLEAN;

  begin
    --**PUT_LINE ("*****enter PROCESS_QUERY_SPECIFICATION_FOR_INSERT");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
      if TOKEN.ID.all = "SELEC" then
        SELECT_TYPE := SELEC.SELEC;
      elsif TOKEN.ID.all = "SELECT_ALL" then
        SELECT_TYPE := SELEC.SELECT_ALL;
      elsif TOKEN.ID.all = "SELECT_DISTINCT" then
        SELECT_TYPE := SELEC.SELECT_DISTINCT;
      else
        --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_INSERT");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
      end if;
    else
      --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_INSERT");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting SELEC, SELECT_ALL or SELECT_DISTINCT");
    end if;
```

```
    SYNTACTICALLY.SKIP_SELECT_CLAUSE;
    TABLE_EXPRESSION.PROCESS_FROM_CLAUSE (FROM_INFO);
    LEXICAL_ANALYZER.RESTORE_SKIPPED_TOKENS;
    PROCESS_SELECT_LIST_OR_STAR (FROM_INFO, SELECTED_ITEMS, SELECT_STAR);
    COMPARE_SELECT_ITEMS_AND_COLUMN_LIST (SELECTED_ITEMS, COLUMN_LIST, TOKEN);
    TABLE_EXPRESSION.PROCESS_REST_OF_TABLE_EXPRESSION (FROM_INFO);
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_INSERT");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting right parenthesis");
    end if;
    if SELECT_STAR then
      SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.STAR, SELEC.INSERT_ITEM, null);
    elsif SELECTED_ITEMS.NEXT_ITEM = null and then
          SELECTED_ITEMS.RESULT_DESCRIPTOR.LOCATION = IN_PROGRAM then
      if SELECTED_ITEMS.RESULT_DESCRIPTOR.TYPE_IS = IS_KNOWN then
        SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE, SELEC.INSERT_ITEM,
                    SELECTED_ITEMS.RESULT_DESCRIPTOR.KNOWN_TYPE.FULL_NAME);
      else
        SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.PROGRAM_VALUE, SELEC.INSERT_ITEM,
                            SELECTED_ITEMS.STRONGLY_TYPED_DES.FULL_NAME);
      end if;
    else
      SELEC.REQUIRED_FOR (SELECT_TYPE, SELEC.SQL_OBJECT, SELEC.INSERT_ITEM,
                          null);
    end if;
    --**PUT_LINE ("*****exit PROCESS_QUERY_SPECIFICATION_FOR_INSERT");
  end PROCESS_QUERY_SPECIFICATION_FOR_INSERT;

----------------------------------------------------------------------
--   PROCESS_INSERT_VALUE_LIST          insert_value [ AND insert_value] ..
--                                      insert_value
--                                            value_specification

  procedure PROCESS_INSERT_VALUE_LIST
           (COLUMN_LIST : LIST_OF_COLUMNS;
            FROM_INFO   : FROM_CLAUSE.INFORMATION) is

    TENTATIVE_FUNCTIONS : TENTATIVE.FUNCTION_LIST;
    VALUE_RESULT_DES    : RESULT.DESCRIPTOR;
    THIS_COLUMN         : LIST_OF_COLUMNS := COLUMN_LIST;
    TOKEN               : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    RETURN_TYPE         : RESULT.DESCRIPTOR;
    COMPARABLE          : RESULT.COMPARABILITY;
    PARM                : GENERATED_FUNCTIONS.OPERAND_KIND;
```

```
    PARM_DES              : DDL_DEFINITIONS.ACCESS_FULL_NAME_DESCRIPTOR;
    PARM_TYPE             : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    COUNT                 : NATURAL := 0;

  begin
    --**PUT_LINE ("*****enter PROCESS_INSERT_VALUE_LIST");
    loop
      COUNT := COUNT + 1;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      EXPRESSION.PROCESS_VALUE_EXPRESSION (FROM_INFO, TRUE,
                 SEMANTICALLY.PROGRAM_VALUE, TENTATIVE_FUNCTIONS,
                 VALUE_RESULT_DES);
      if VALUE_RESULT_DES.LOCATION = RESULT.IN_PROGRAM then
        PARM := GENERATED_FUNCTIONS.O_USER_TYPE;
        PARM_TYPE := SEMANTICALLY.STRONGLY_TYPE (VALUE_RESULT_DES);
        PARM_DES := PARM_TYPE.FULL_NAME;
      else
        PARM := GENERATED_FUNCTIONS.O_SQL_OBJECT;
        PARM_DES := null;
        TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT (TENTATIVE_FUNCTIONS);
      end if;
      RESULT.COMBINED_TYPE (THIS_COLUMN.COLUMN_DES.BASE_TYPE,
             VALUE_RESULT_DES, RETURN_TYPE, COMPARABLE);
      if COMPARABLE = RESULT.IS_NOT_COMPARABLE then
        LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
        "Insert value is not comparable with insert column list");
      end if;
      if VALUE_RESULT_DES.LOCATION = RESULT.IN_PROGRAM then
        PARM := GENERATED_FUNCTIONS.O_USER_TYPE;
      else
        PARM := GENERATED_FUNCTIONS.O_SQL_OBJECT;
      end if;
      if COUNT = 1 then
        GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
          (ADA_SQL_FUNCTION_DEFINITIONS.O_LE,
           GENERATED_FUNCTIONS.O_INSERT_ITEM, null,
           PARM, PARM_DES, GENERATED_FUNCTIONS.O_INSERT_ITEM, null);
      else
        GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
          (ADA_SQL_FUNCTION_DEFINITIONS.O_AND,
           GENERATED_FUNCTIONS.O_INSERT_ITEM, null,
           PARM, PARM_DES, GENERATED_FUNCTIONS.O_INSERT_ITEM, null);
      end if;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      exit when TOKEN.KIND /= LEXICAL_ANALYZER.RESERVED_WORD or else
         TOKEN.RESERVED_WORD /= LEXICAL_ANALYZER.R_AND;
      THIS_COLUMN := THIS_COLUMN.NEXT_COLUMN;
      if THIS_COLUMN = null then
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
```

```
        ( TOKEN , "More values than columns" );
      end if;
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    end loop;
    if THIS_COLUMN.NEXT_COLUMN /= null then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
      ( TOKEN , "Not enough values supplied for all columns" );
    end if;
    --**PUT_LINE ("*****exit PROCESS_INSERT_VALUE_LIST");
  end PROCESS_INSERT_VALUE_LIST;

------------------------------------------------------------------------
--   NEW_COLUMN_ALREADY_IN_LIST

  function NEW_COLUMN_ALREADY_IN_LIST
          (NEW_COL   : LIST_OF_COLUMNS;
           FIRST_COL : LIST_OF_COLUMNS)
           return      BOOLEAN is

    COL : LIST_OF_COLUMNS := FIRST_COL;

  begin
    --**PUT_LINE ("*****enter NEW_COLUMN_ALREADY_IN_LIST");
    while COL /= null loop
      if COL.COLUMN_DES = NEW_COL.COLUMN_DES then
        --**PUT_LINE ("*****exit NEW_COLUMN_ALREADY_IN_LIST");
        return TRUE;
      end if;
      COL := COL.NEXT_COLUMN;
    end loop;
    --**PUT_LINE ("*****exit NEW_COLUMN_ALREADY_IN_LIST");
    return FALSE;
  end NEW_COLUMN_ALREADY_IN_LIST;

--------------------------------------------------------------------
--  PROCESS_INSERT_COLUMN_LIST            column_name [ & column_name ] ...

  procedure PROCESS_INSERT_COLUMN_LIST
           (FROM_INFO   : FROM_CLAUSE.INFORMATION;
            TABLE_DES   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            COLUMN_LIST : out LIST_OF_COLUMNS) is

    COL_DES     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                  TABLE_DES.FIRST_COMPONENT;
    FIRST_COL   : LIST_OF_COLUMNS := null;
    LAST_COL    : LIST_OF_COLUMNS := null;
    NEW_COL     : LIST_OF_COLUMNS := null;
    TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    NAME_INFO   : NAME.INFORMATION;
```

```
begin
  --**PUT_LINE ("*****enter PROCESS_INSERT_COLUMN_LIST");
  loop
    NAME_INFO := NAME.AT_CURRENT_INPUT_POINT (FROM_INFO,
                 NAME.IS_COLUMN_NAME, TRUE, FALSE);
    NEW_COL := new LIST_OF_COLUMNS_RECORD '
               (COLUMN_DES => NAME_INFO.UNQUALIFIED_COLUMN.TYPE_IS,
                NEXT_COLUMN => null);
    if NEW_COLUMN_ALREADY_IN_LIST (NEW_COL, FIRST_COL) then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR (TOKEN,
      "Column name used more than once");
    end if;
    if FIRST_COL = null then
      FIRST_COL := NEW_COL;
      LAST_COL := NEW_COL;
    else
      LAST_COL.NEXT_COLUMN := NEW_COL;
      LAST_COL := NEW_COL;
    end if;
    UNQUALIFIED_NAME.RETURNS_SQL_OBJECT (NEW_COL.COLUMN_DES.FULL_NAME.NAME);
    SYNTACTICALLY.GOBBLE_NAME (NAME_INFO);
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.AMPERSAND then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
          (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND,
           GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
           GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
           GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
    else
      exit;
    end if;
  end loop;
  COLUMN_LIST := FIRST_COL;
  --**PUT_LINE ("*****exit PROCESS_INSERT_COLUMN_LIST");
end PROCESS_INSERT_COLUMN_LIST;

------------------------------------------------------------------
--  SET_COLUMN_LIST_FOR_ALL

  procedure SET_COLUMN_LIST_FOR_ALL
          (TABLE_DES   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
           COLUMN_LIST : out LIST_OF_COLUMNS) is

    COL_DES     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
                  TABLE_DES.FIRST_COMPONENT;
    FIRST_COL   : LIST_OF_COLUMNS := null;
    LAST_COL    : LIST_OF_COLUMNS := null;
```

```
    NEW_COL     : LIST_OF_COLUMNS := null;

  begin
    --**PUT_LINE ("*****enter SET_COLUMN_LIST_FOR_ALL");
    while COL_DES /= null loop
      NEW_COL := new LIST_OF_COLUMNS_RECORD '
                    (COLUMN_DES => COL_DES,
                     NEXT_COLUMN => null);
      if FIRST_COL = null then
        FIRST_COL := NEW_COL;
        LAST_COL := NEW_COL;
      else
        LAST_COL.NEXT_COLUMN := NEW_COL;
        LAST_COL := NEW_COL;
      end if;
    COL_DES := COL_DES.NEXT_ONE;
    end loop;
    COLUMN_LIST := FIRST_COL;
    --**PUT_LINE ("*****exit SET_COLUMN_LIST_FOR_ALL");
  end SET_COLUMN_LIST_FOR_ALL;

------------------------------------------------------------------
--  PROCESS_TABLE_NAME

  procedure PROCESS_TABLE_NAME
           (TABLE_DES : out DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
            FROM_INFO : out FROM_CLAUSE.INFORMATION) is

    TOKEN           : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    TABLE_STATUS    : TABLE.NAME_STATUS;
    TABLE_DESC      : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR := null;
    TABLE_FROM_INFO : FROM_CLAUSE.INFORMATION :=
                      FROM_CLAUSE.AT_NEW_SCOPE (null);

  begin
    --**PUT_LINE ("*****enter PROCESS_TABLE_NAME");
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
      TABLE.DESCRIPTOR_FOR (TOKEN.ID.all, TABLE_STATUS, TABLE_DESC);
      case TABLE_STATUS is
        when TABLE.NAME_UNDEFINED =>
                   --**PUT_LINE ("*****exit PROCESS_TABLE_NAME");
                   LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                   "Table name is undefined");
        when TABLE.NAME_AMBIGUOUS =>
                   --**PUT_LINE ("*****exit PROCESS_TABLE_NAME");
                   LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
                   "Table name is ambiguous");
        when TABLE.NAME_UNIQUE    =>
```

```
                LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        end case;
        TABLE_FROM_INFO := FROM_CLAUSE.AT_NEW_SCOPE (null);
        FROM_CLAUSE.NAMES_EXPOSED_TABLE (TABLE_FROM_INFO, TABLE_DESC);
        UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DESC.FULL_NAME.NAME);
      else
        --**PUT_LINE ("*****exit PROCESS_TABLE_NAME");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting table name");
      end if;
      TABLE_DES := TABLE_DESC;
      FROM_INFO := TABLE_FROM_INFO;
      --**PUT_LINE ("*****exit PROCESS_TABLE_NAME");
    end PROCESS_TABLE_NAME;

  -------------------------------------------------------------------------
  -- PROCESS_INSERT_INTO           INSERT_INTO ( table_name
  --                                 [ ( insert_column_list ) ] ,
  --                                 VALUES <= insert_value_list
  --                               | query_specification );

    procedure PROCESS_INSERT_INTO is

      TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
      FROM_INFO   : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE (null);
      RESULT_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      TABLE_DES   : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
      COLUMN_LIST : LIST_OF_COLUMNS;

    begin
      --**PUT_LINE ("*****enter PROCESS_INSERT_INTO");
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
         TOKEN.ID.all = "INSERT_INTO" then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting INSERT_INTO");
      end if;
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
        "Expecting left parenthesis");
      end if;
```

```
PROCESS_TABLE_NAME (TABLE_DES, FROM_INFO);
TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
   TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  PROCESS_INSERT_COLUMN_LIST (FROM_INFO, TABLE_DES, COLUMN_LIST);
  UNQUALIFIED_NAME.RETURNS_TABLE_NAME_WITH_COLUMN_LIST
                  (TABLE_DES.FULL_NAME.NAME);
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
     TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  else
    --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
    "Expecting right parenthesis");
  end if;
else
  SET_COLUMN_LIST_FOR_ALL (TABLE_DES, COLUMN_LIST);
  UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DES.FULL_NAME.NAME);
end if;
if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
   TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA then
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
else
  --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
  LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
  "Expecting comma");
end if;
TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER and then
   TOKEN.ID.all = "VALUES" then
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
     TOKEN.DELIMITER = LEXICAL_ANALYZER.LESS_THAN_OR_EQUAL then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    PROCESS_INSERT_VALUE_LIST (COLUMN_LIST, FROM_INFO);
    PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.VALUES_FUNCTION);
  else
    --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
    "Expecting <=");
  end if;
else
  PROCESS_QUERY_SPECIFICATION_FOR_INSERT (COLUMN_LIST);
end if;
TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
```

```
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting right parenthesis");
    end if;
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       TOKEN.DELIMITER = LEXICAL_ANALYZER.SEMICOLON then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    else
      --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN,
      "Expecting semicolon");
    end if;
    PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.INSERT_INTO_PROCEDURE);
    --**PUT_LINE ("*****exit PROCESS_INSERT_INTO");
  end PROCESS_INSERT_INTO;

end SELECT_STATEMENT;
```

3.11.81 package statementb.ada

```
with LEXICAL_ANALYZER, NAME, PREDEFINED_TYPE, PREDEFINED, TABLE,
     DDL_DEFINITIONS, FROM_CLAUSE, UNQUALIFIED_NAME, SEARCH_CONDITION,
     SYNTACTICALLY, COLUMN_LIST, EXPRESSION, SEMANTICALLY, TENTATIVE,
     GENERATED_FUNCTIONS, RESULT, ADA_SQL_FUNCTION_DEFINITIONS, CORRELATION;
use NAME, LEXICAL_ANALYZER, DDL_DEFINITIONS;
package body STATEMENT is

------------------------------------------------------------------------
-- VALID_CURSOR_NAME - validate that the CURSOR_NAME is a variable in
-- name.at_current_input_point, make sure the variable is of cursor name type
-- call predefined.text_required_for (predefined.open_procedure)

  function VALID_CURSOR_NAME
          (OPEN_CLOSE  : STRING;
           CURSOR_NAME : STRING)
           return        BOOLEAN is

    NAME_INFO : NAME.INFORMATION;

  begin
    NAME_INFO := NAME.AT_CURRENT_INPUT_POINT (null, NAME.IS_PROGRAM_VARIABLE,
                 TRUE, FALSE, FALSE);
    SYNTACTICALLY.GOBBLE_NAME (NAME_INFO);
    if NAME_INFO.KIND = NAME.OF_VARIABLE and then
       NAME_INFO.VARIABLE_TYPE.TYPE_IS.ULT_PARENT_TYPE =
```

```
          PREDEFINED_TYPE.CURSOR_DEFINITION.CURSOR_NAME then
    if OPEN_CLOSE = "OPEN" then
      PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.OPEN_PROCEDURE);
    elsif OPEN_CLOSE = "CLOSE" then
      PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.CLOSE_PROCEDURE);
    end if;
    return TRUE;
  else
    return FALSE;
  end if;
exception
  when LEXICAL_ANALYZER.SYNTAX_ERROR => return FALSE;
end VALID_CURSOR_NAME;

-------------------------------------------------------------------
--  PROCESS_OPEN_OR_CLOSE_STATEMENT       open_or_close_statement
--                                            OPEN | CLOSE ( cursor_name ) ;

  procedure PROCESS_OPEN_OR_CLOSE_STATEMENT
           (OPEN_CLOSE : STRING) is
  begin
    SYNTACTICALLY.PROCESS_KEYWORD (OPEN_CLOSE);
    if SYNTACTICALLY.IS_DELIMITER (LEXICAL_ANALYZER.LEFT_PARENTHESIS) then
      SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.LEFT_PARENTHESIS);
      if SYNTACTICALLY.IS_IDENTIFIER and then
         VALID_CURSOR_NAME
           (OPEN_CLOSE, LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN.ID.all) then
        SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
        SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.SEMICOLON);
      end if;
    end if;
  end PROCESS_OPEN_OR_CLOSE_STATEMENT;

-------------------------------------------------------------------
--  PROCESS_OPEN_STATEMENT             open_statement
--                                            OPEN ( cursor_name ) ;

  procedure PROCESS_OPEN_STATEMENT is
  begin
    PROCESS_OPEN_OR_CLOSE_STATEMENT ("OPEN");
  end PROCESS_OPEN_STATEMENT;

-------------------------------------------------------------------
--  PROCESS_DELETE_STATEMENT_SEARCHED     delete_statement_searched
--                                          DELETE_FROM ( table_name [,
--                                          WHERE => search_condition ] ) ;

  procedure PROCESS_DELETE_STATEMENT_SEARCHED is
```

```
    TOKEN            : LEXICAL_ANALYZER.LEXICAL_TOKEN;
    TABLE_STATUS     : TABLE.NAME_STATUS;
    TABLE_DES        : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
    TABLE_FROM_INFO  : FROM_CLAUSE.INFORMATION;

  begin
    SYNTACTICALLY.PROCESS_KEYWORD ("DELETE_FROM");
    SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.LEFT_PARENTHESIS);
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if SYNTACTICALLY.IS_IDENTIFIER then
      TABLE.DESCRIPTOR_FOR (TOKEN.ID.all, TABLE_STATUS, TABLE_DES);
      case TABLE_STATUS is
        when TABLE.NAME_UNDEFINED =>
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
            (TOKEN, "Table name is undefined");
        when TABLE.NAME_AMBIGUOUS =>
           LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
             (TOKEN, "Table name is ambiguous");
        when TABLE.NAME_UNIQUE    =>
           LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      end case;
      TABLE_FROM_INFO := FROM_CLAUSE.AT_NEW_SCOPE (null);
      FROM_CLAUSE.NAMES_EXPOSED_TABLE (TABLE_FROM_INFO, TABLE_DES);
      UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DES.FULL_NAME.NAME);
    else
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, "Expecting table name");
    end if;
    if SYNTACTICALLY.IS_DELIMITER (LEXICAL_ANALYZER.COMMA) then
      SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.COMMA);
      SYNTACTICALLY.PROCESS_KEYWORD ("WHERE");
      SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.ARROW);
      SEARCH_CONDITION.PROCESS_SEARCH_CONDITION (TABLE_FROM_INFO);
    end if;
    SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
    SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.SEMICOLON);
    PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.DELETE_SEARCHED_PROCEDURE);
  end PROCESS_DELETE_STATEMENT_SEARCHED;

------------------------------------------------------------------------
-- PROCESS_SET_CLAUSE                        set_clause
--                                              object_column <= value_expression
--                                              [ AND ....]

  procedure PROCESS_SET_CLAUSE
           (TABLE_FROM_INFO : FROM_CLAUSE.INFORMATION) is

    TOKEN                 : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    COLUMN_INFO           : NAME.INFORMATION;
    TENTATIVE_FUNCTIONS   : TENTATIVE.FUNCTION_LIST;
```

```
    VALUE_RESULT_DES     : RESULT.DESCRIPTOR;
    RETURN_TYPE          : RESULT.DESCRIPTOR;
    COMPARABLE           : RESULT.COMPARABILITY;
    COLUMN_RESULT_DES    : RESULT.DESCRIPTOR (RESULT.IS_KNOWN);
    LIST_OF_COLUMNS      : COLUMN_LIST.ELEMENT := null;

  begin
    loop
      COLUMN_INFO := NAME.AT_CURRENT_INPUT_POINT (TABLE_FROM_INFO,
                     NAME.IS_COLUMN_NAME, TRUE, FALSE);
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      COLUMN_LIST.ADD_NEW_COLUMN (LIST_OF_COLUMNS,
                          COLUMN_INFO.UNQUALIFIED_COLUMN, TOKEN);
      SYNTACTICALLY.GOBBLE_NAME (COLUMN_INFO);
      UNQUALIFIED_NAME.RETURNS_TYPED_RESULT
                  (COLUMN_INFO.UNQUALIFIED_COLUMN.NAME,
                   COLUMN_INFO.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE.FULL_NAME);
      TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.LESS_THAN_OR_EQUAL);
      EXPRESSION.PROCESS_VALUE_EXPRESSION (TABLE_FROM_INFO, TRUE,
                  SEMANTICALLY.ANY_VALUE, TENTATIVE_FUNCTIONS,
                  VALUE_RESULT_DES);
      TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED (TENTATIVE_FUNCTIONS,
                  COLUMN_INFO.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE);
      SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS (TOKEN,
                  COLUMN_INFO.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE,
                  VALUE_RESULT_DES, RETURN_TYPE, COMPARABLE);
      COLUMN_RESULT_DES.KNOWN_TYPE :=
       COLUMN_INFO.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE;
      COLUMN_RESULT_DES.LOCATION := RESULT.IN_DATABASE;
      SEMANTICALLY.MAKE_BINARY_OPERATION (ADA_SQL_FUNCTION_DEFINITIONS.O_LE,
                  COLUMN_INFO.UNQUALIFIED_COLUMN.TYPE_IS.BASE_TYPE,
                  COLUMN_RESULT_DES, VALUE_RESULT_DES,
                  GENERATED_FUNCTIONS.O_SQL_OBJECT);
    exit when NOT SYNTACTICALLY.IS_RESERVED_WORD (LEXICAL_ANALYZER.R_AND);
      SYNTACTICALLY.PROCESS_RESERVED_WORD (LEXICAL_ANALYZER.R_AND);
      GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
                (ADA_SQL_FUNCTION_DEFINITIONS.O_AND,
                 GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
                 GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
                 GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
    end loop;
  end PROCESS_SET_CLAUSE;

------------------------------------------------------------------
--  PROCESS_UPDATE_STATEMENT_SEARCHED   update_statement_searched
--                                      UPDATE ( table_name,
--                                        SET => set_clause
--                                        [ AND set_clause ... ]
```

```
--                                        [, WHERE => search_condition ] ) ;

 procedure PROCESS_UPDATE_STATEMENT_SEARCHED is

   TOKEN            : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
   TABLE_STATUS     : TABLE.NAME_STATUS;
   TABLE_DES        : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
   TABLE_FROM_INFO  : FROM_CLAUSE.INFORMATION;

 begin
   SYNTACTICALLY.PROCESS_KEYWORD ("UPDATE");
   SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.LEFT_PARENTHESIS);
   TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
   if SYNTACTICALLY.IS_IDENTIFIER then
     TABLE.DESCRIPTOR_FOR (TOKEN.ID.all, TABLE_STATUS, TABLE_DES);
     case TABLE_STATUS is
       when TABLE.NAME_UNDEFINED =>
         LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
           (TOKEN, "Table name is undefined");
       when TABLE.NAME_AMBIGUOUS =>
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
            (TOKEN, "Table name is ambiguous");
       when TABLE.NAME_UNIQUE    =>
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
     end case;
     TABLE_FROM_INFO := FROM_CLAUSE.AT_NEW_SCOPE (null);
     FROM_CLAUSE.NAMES_EXPOSED_TABLE (TABLE_FROM_INFO, TABLE_DES);
     UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DES.FULL_NAME.NAME);
   else
     LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TOKEN, "Expecting table name");
   end if;
   SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.COMMA);
   SYNTACTICALLY.PROCESS_KEYWORD ("SET");
   SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.ARROW);
   PROCESS_SET_CLAUSE (TABLE_FROM_INFO);
   if SYNTACTICALLY.IS_DELIMITER (LEXICAL_ANALYZER.COMMA) then
     SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.COMMA);
     SYNTACTICALLY.PROCESS_KEYWORD ("WHERE");
     SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.ARROW);
     SEARCH_CONDITION.PROCESS_SEARCH_CONDITION (TABLE_FROM_INFO);
   end if;
   SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
   SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.SEMICOLON);
   PREDEFINED.TEXT_REQUIRED_FOR (PREDEFINED.UPDATE_SEARCHED_PROCEDURE);
 end PROCESS_UPDATE_STATEMENT_SEARCHED;

------------------------------------------------------------------
--  PROCESS_CLOSE_STATEMENT              close_statement
--                                           CLOSE ( cursor_name ) ;
```

```
procedure PROCESS_CLOSE_STATEMENT is
begin
  PROCESS_OPEN_OR_CLOSE_STATEMENT ("CLOSE");
end PROCESS_CLOSE_STATEMENT;

function UPPER_CASE
   (S : STRING) return STRING is
   RESULT : STRING (S'RANGE) := S;
begin
   for I in RESULT'RANGE loop
      if RESULT(I) in 'a'..'z' then
         RESULT(I) := CHARACTER'VAL (CHARACTER'POS (RESULT(I)) - 32);
      end if;
   end loop;
   return RESULT;
end UPPER_CASE;

-----------------------------------------------------------------------
-- PROCESS_PACKAGE     PACKAGE package_name IS NEW
--                             table_name_CORRELATION.NAME ("package_name");

procedure PROCESS_PACKAGE is

  TOKEN              : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  PACKAGE_TOKEN      : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  TABLE_TOKEN        : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  STRING_TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  TABLE_STATUS       : TABLE.NAME_STATUS;
  TABLE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
  PACKAGE_STATUS       : TABLE.NAME_STATUS;
  PACKAGE_DESCRIPTOR : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;

begin
  -- return as soon as we determine that this is not a correlation name
  -- package declaration.

  -- verify PACKAGE
  SYNTACTICALLY.PROCESS_RESERVED_WORD (LEXICAL_ANALYZER.R_PACKAGE);
  -- verify package name is identifier
  if not SYNTACTICALLY.IS_IDENTIFIER then
    return;
  end if;
  PACKAGE_TOKEN := LEXICAL_ANALYZER.NEXT_TOKEN;
  -- verify IS
  if not SYNTACTICALLY.IS_RESERVED_WORD (LEXICAL_ANALYZER.R_IS) then
    return;
  end if;
  SYNTACTICALLY.PROCESS_RESERVED_WORD (LEXICAL_ANALYZER.R_IS);
  -- verify NEW
```

```
if not SYNTACTICALLY.IS_RESERVED_WORD (LEXICAL_ANALYZER.R_NEW) then
  return;
end if;
SYNTACTICALLY.PROCESS_RESERVED_WORD (LEXICAL_ANALYZER.R_NEW);
-- verify table name is identifier with _CORRELATION on end
if not SYNTACTICALLY.IS_IDENTIFIER then
  return;
end if;
TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
if TOKEN.ID.all'LENGTH <= 12 or else
   TOKEN.ID.all (TOKEN.ID.all'LAST-11..TOKEN.ID.all'LAST) /= "_CORRELATION"
then
  return;
end if;
TABLE_TOKEN := LEXICAL_ANALYZER.NEXT_TOKEN;
-- verify .
if not SYNTACTICALLY.IS_DELIMITER (LEXICAL_ANALYZER.DOT) then
  return;
end if;
SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.DOT);
-- verify NAME
if not SYNTACTICALLY.IS_IDENTIFIER or else
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN.ID.all /= "NAME" then
  return;
end if;
LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
-- if we get this far, it is pretty obvious that this is a correlation
-- package declaration so start issuing syntax diagnostics (if necessary)
SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.LEFT_PARENTHESIS);
STRING_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
if STRING_TOKEN.KIND = LEXICAL_ANALYZER.STRING_LITERAL and then
   UPPER_CASE (STRING_TOKEN.STRING_IMAGE.all) = PACKAGE_TOKEN.ID.all then
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
  SYNTACTICALLY.PROCESS_DELIMITER (LEXICAL_ANALYZER.SEMICOLON);
else
  LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
    (STRING_TOKEN, "Expecting """ & PACKAGE_TOKEN.ID.all & """");
end if;
if NAME.IS_PACKAGE_WITHED (PACKAGE_TOKEN.ID.all) then
  LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (PACKAGE_TOKEN,
  "Package name conflicts with withed package");
end if;
TABLE.DESCRIPTOR_FOR
   (TABLE_TOKEN.ID.all   -- strip off trailing "_CORRELATION"
       (TABLE_TOKEN.ID.all'FIRST..TABLE_TOKEN.ID.all'LAST-12),
    TABLE_STATUS,
    TABLE_DESCRIPTOR);
case TABLE_STATUS is
```

```
        when TABLE.NAME_UNDEFINED =>
             LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
             "Table name is undefined");
        when TABLE.NAME_AMBIGUOUS =>
             LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
             "Table name is ambiguous");
        when TABLE.NAME_UNIQUE =>
             null;
      end case;
      TABLE.DESCRIPTOR_FOR (PACKAGE_TOKEN.ID.all, PACKAGE_STATUS,
                            PACKAGE_DESCRIPTOR);
      case PACKAGE_STATUS is
        when TABLE.NAME_UNIQUE | TABLE.NAME_AMBIGUOUS =>
             LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (PACKAGE_TOKEN,
             "Correlation name duplicates a table name");
        when TABLE.NAME_UNDEFINED =>
             null;
      end case;
      if not CORRELATION.NAME_DECLARATION_IS_VALID (PACKAGE_TOKEN.ID.all,
                  TABLE_DESCRIPTOR) then
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (PACKAGE_TOKEN,
        "Correlation name is duplicated");
      end if;
    end PROCESS_PACKAGE;
  end STATEMENT;
```

3.11.82 package searchb.ada

```
-- searchb.ada - routine to process a search condition

with ADA_SQL_FUNCTION_DEFINITIONS, DDL_DEFINITIONS, EXPRESSION, FROM_CLAUSE,
 GENERATED_FUNCTIONS, LEXICAL_ANALYZER, PREDEFINED_TYPE, RESULT, SELEC,
 SEMANTICALLY, SYNTACTICALLY, TABLE_EXPRESSION, TENTATIVE;
 use DDL_DEFINITIONS, LEXICAL_ANALYZER, RESULT;
package body SEARCH_CONDITION is

  type BOOLEAN_OPERATIONS is array ( LEXICAL_ANALYZER.RESERVED_WORD_KIND ) of
   ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;

  BOOLEAN_OPERATION : constant BOOLEAN_OPERATIONS :=
  ( ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ABORT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ABS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ACCEPT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ACCESS
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ALL
    ADA_SQL_FUNCTION_DEFINITIONS.O_AND    ,    -- R_AND
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ARRAY
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_AT
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_BEGIN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_BODY
```

```
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_CASE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_CONSTANT
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_DECLARE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_DELAY
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_DELTA
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_DIGITS
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_DO
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ELSE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ELSIF
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_END
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_ENTRY
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_EXCEPTION
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_EXIT
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_FOR
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_FUNCTION
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_GENERIC
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_GOTO
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_IF
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_IN
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_IS
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_LIMITED
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_LOOP
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_MOD
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_NEW
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_NOT
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_NULL
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_OF
ADA_SQL_FUNCTION_DEFINITIONS.O_OR     ,    -- R_OR
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_OTHERS
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_OUT
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_PACKAGE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_PRAGMA
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_PRIVATE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_PROCEDURE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_RAISE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_RANGE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_RECORD
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_REM
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_RENAMES
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_RETURN
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_REVERSE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_SELECT
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_SEPARATE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_SUBTYPE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_TASK
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_TERMINATE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_THEN
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_TYPE
ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_USE
```

```
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_WHEN
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_WHILE
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP,    -- R_WITH
    ADA_SQL_FUNCTION_DEFINITIONS.O_NULL_OP ); -- R_XOR

  type PREDICATE_WORDS is ( EQ , NE , BETWEEN , IS_IN , NOT_IN , LIKE );
  -- null predicate, quantified predicate, and exists predicate not impl. now

  type PREDICATE_OPERATIONS is array ( PREDICATE_WORDS )
   of ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;

  PREDICATE_OPERATION : constant PREDICATE_OPERATIONS :=
  ( EQ      => ADA_SQL_FUNCTION_DEFINITIONS.O_EQ,
    NE      => ADA_SQL_FUNCTION_DEFINITIONS.O_NE,
    BETWEEN => ADA_SQL_FUNCTION_DEFINITIONS.O_BETWEEN,
    IS_IN   => ADA_SQL_FUNCTION_DEFINITIONS.O_IS_IN,
    NOT_IN  => ADA_SQL_FUNCTION_DEFINITIONS.O_NOT_IN,
    LIKE    => ADA_SQL_FUNCTION_DEFINITIONS.O_LIKE );

  SELEC_PARAMETER_KIND : constant array ( RESULT.VALUE_LOCATION )
   of SELEC.PARAMETER_TYPE :=
  ( RESULT.IN_PROGRAM  => SELEC.PROGRAM_VALUE,
    RESULT.IN_DATABASE => SELEC.DATABASE_VALUE );

  procedure PROCESS_STAR_SUBQUERY
            ( TOKEN                : in  LEXICAL_ANALYZER.LEXICAL_TOKEN;
              SELECT_TYPE          : in  SELEC.ROUTINE_NAME;
              FROM                 : in  FROM_CLAUSE.INFORMATION;
              TENTATIVE_FUNCTIONS  : out TENTATIVE.FUNCTION_LIST;
              RETURN_TYPE          : out RESULT.DESCRIPTOR ) is
    T1                   : TENTATIVE.FUNCTION_LIST;
    R1                   : RESULT.DESCRIPTOR;
    TABLE                : FROM_CLAUSE.TABLE_ENTRY;
    MORE_THAN_ONE_TABLE  : BOOLEAN;
    TABLE_DESCRIPTOR     : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
  begin
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    TABLE := FROM_CLAUSE.TABLES_AT_CURRENT_SCOPE ( FROM );
    FROM_CLAUSE.NEXT_TABLE
    ( TABLE , MORE_THAN_ONE_TABLE , TABLE_DESCRIPTOR );
    if MORE_THAN_ONE_TABLE or else
     TABLE_DESCRIPTOR.FIRST_COMPONENT.NEXT_ONE /= null then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
      ( TOKEN , "Only one column may be selected in a subquery" );
    end if;
    T1 := TENTATIVE.FUNCTION_LIST_CREATOR;
    R1 :=
    ( TYPE_IS     => RESULT.IS_KNOWN,
      LOCATION    => RESULT.IN_DATABASE,
```

```
    KNOWN_TYPE => TABLE_DESCRIPTOR.FIRST_COMPONENT.BASE_TYPE );
  TENTATIVE.FUNCTION_REQUIRED_FOR_SELECT_FUNCTION
  ( T1 , R1 , SELECT_TYPE , SELEC.STAR );
  TENTATIVE_FUNCTIONS := T1;
  RETURN_TYPE := R1;
end PROCESS_STAR_SUBQUERY;

procedure PROCESS_SUBQUERY
          ( SELECT_TYPE         : in  SELEC.ROUTINE_NAME;
            FROM                : in  FROM_CLAUSE.INFORMATION;
            TENTATIVE_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RETURN_TYPE         : out RESULT.DESCRIPTOR ) is
  OUR_FROM : FROM_CLAUSE.INFORMATION := FROM_CLAUSE.AT_NEW_SCOPE ( FROM );
  TOKEN    : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  T1       : TENTATIVE.FUNCTION_LIST;
  R1       : RESULT.DESCRIPTOR;
begin
  SYNTACTICALLY.SKIP_SELECT_CLAUSE;
  TABLE_EXPRESSION.PROCESS_FROM_CLAUSE ( OUR_FROM );
  LEXICAL_ANALYZER.RESTORE_SKIPPED_TOKENS;
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND = LEXICAL_ANALYZER.CHARACTER_LITERAL and then
   TOKEN.CHARACTER_VALUE = '*' then
    PROCESS_STAR_SUBQUERY
    ( TOKEN , SELECT_TYPE , OUR_FROM , TENTATIVE_FUNCTIONS , RETURN_TYPE );
  else
    EXPRESSION.PROCESS_VALUE_EXPRESSION
    ( OUR_FROM , TRUE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
    TENTATIVE.FUNCTION_REQUIRED_FOR_SELECT_FUNCTION
    ( T1 , R1 , SELECT_TYPE , SELEC_PARAMETER_KIND ( R1.LOCATION ) );
    R1.LOCATION := RESULT.IN_DATABASE;
    TENTATIVE_FUNCTIONS := T1;
    RETURN_TYPE := R1;
  end if;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.COMMA );
  TABLE_EXPRESSION.PROCESS_REST_OF_TABLE_EXPRESSION ( OUR_FROM );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
end PROCESS_SUBQUERY;

procedure VALIDATE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
          ( TOKEN              : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            OPERATION          : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
            LEFT_FUNCTIONS,
            RIGHT_FUNCTIONS    : TENTATIVE.FUNCTION_LIST;
            LEFT_TYPE,
            RIGHT_TYPE,
            COMBINED_TYPE      : RESULT.DESCRIPTOR;
            RETURN_TYPE        : GENERATED_FUNCTIONS.OPERAND_KIND ) is
  STRONG_TYPE : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR :=
```

```
   SEMANTICALLY.VALIDATE_STRONGLY_TYPED ( TOKEN , COMBINED_TYPE );
begin
  if STRONG_TYPE /= null then
    TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED
    ( LEFT_FUNCTIONS , STRONG_TYPE );
    TENTATIVE.FUNCTIONS_RETURN_STRONGLY_TYPED
    ( RIGHT_FUNCTIONS , STRONG_TYPE );
    SEMANTICALLY.MAKE_BINARY_OPERATION
    ( OPERATION , STRONG_TYPE , LEFT_TYPE , RIGHT_TYPE , RETURN_TYPE );
  end if;
end VALIDATE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION;

procedure VALIDATE_COMPARABLE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
          ( TOKEN             : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            OPERATION         : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
            LEFT_FUNCTIONS,
            RIGHT_FUNCTIONS   : TENTATIVE.FUNCTION_LIST;
            LEFT_TYPE,
            RIGHT_TYPE        : RESULT.DESCRIPTOR;
            RETURN_TYPE       : GENERATED_FUNCTIONS.OPERAND_KIND ) is
  COMBINED_TYPE : RESULT.DESCRIPTOR;
  COMPARABLE    : RESULT.COMPARABILITY;
begin
  SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
  ( TOKEN , LEFT_TYPE , RIGHT_TYPE , COMBINED_TYPE , COMPARABLE );
  if COMPARABLE = RESULT.IS_COMPARABLE then
    VALIDATE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
    ( TOKEN , OPERATION , LEFT_FUNCTIONS , RIGHT_FUNCTIONS , LEFT_TYPE ,
      RIGHT_TYPE , COMBINED_TYPE , RETURN_TYPE );
  end if;
end VALIDATE_COMPARABLE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION;

function VALIDATE_ADA_SQL_VALUE_USED
         ( TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN;
           RETURN_TYPE : RESULT.DESCRIPTOR ) return BOOLEAN is
begin
  if RETURN_TYPE.LOCATION = RESULT.IN_PROGRAM then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN , "Both operands cannot be program values" );
    return FALSE;
  end if;
  return TRUE;
end VALIDATE_ADA_SQL_VALUE_USED;

procedure PROCESS_RIGHT_COMPARISON_OPERAND
          ( OPERATOR_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            OPERATION      : ADA_SQL_FUNCTION_DEFINITIONS.SQL_OPERATION;
            FROM           : FROM_CLAUSE.INFORMATION;
            T1             : TENTATIVE.FUNCTION_LIST;
```

```
            R1              : RESULT.DESCRIPTOR ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  SELECT_TYPE        : SELEC.ROUTINE_NAME;
  DOING_SUBQUERY     : BOOLEAN;
  T2                 : TENTATIVE.FUNCTION_LIST;
  R2, R3             : RESULT.DESCRIPTOR;
  COMPARABLE         : RESULT.COMPARABILITY;
  ERROR              : BOOLEAN;
begin
  SEMANTICALLY.GET_SELECT_WORD ( TOKEN , DOING_SUBQUERY , SELECT_TYPE );
  if DOING_SUBQUERY then
    PROCESS_SUBQUERY ( SELECT_TYPE , FROM , T2, R2 );
  else
    EXPRESSION.PROCESS_VALUE_EXPRESSION
    ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T2 , R2 );
  end if;
  SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
  ( OPERATOR_TOKEN , R1 , R2 , R3 , COMPARABLE );
  if COMPARABLE = RESULT.IS_COMPARABLE and then
   VALIDATE_ADA_SQL_VALUE_USED ( OPERATOR_TOKEN , R3 ) then
    VALIDATE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
    ( OPERATOR_TOKEN , OPERATION , T1 , T2 , R1 , R2 , R3 ,
      GENERATED_FUNCTIONS.O_SQL_OBJECT );
  end if;
end PROCESS_RIGHT_COMPARISON_OPERAND;

procedure PROCESS_PREFIX_COMPARISON
          ( OPERATOR_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            KEYWORD        : PREDICATE_WORDS;
            FROM           : FROM_CLAUSE.INFORMATION ) is
  T1    : TENTATIVE.FUNCTION_LIST;
  R1    : RESULT.DESCRIPTOR;
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.COMMA );
  PROCESS_RIGHT_COMPARISON_OPERAND
  ( OPERATOR_TOKEN , PREDICATE_OPERATION ( KEYWORD ) , FROM , T1 , R1 );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
end PROCESS_PREFIX_COMPARISON;

procedure PROCESS_INFIX_COMPARISON ( FROM : FROM_CLAUSE.INFORMATION ) is
  T1    : TENTATIVE.FUNCTION_LIST;
  R1    : RESULT.DESCRIPTOR;
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  procedure REPORT_ERROR is
```

```
  begin
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR
    ( TOKEN , "Expecting comparison operator" );
  end REPORT_ERROR;
begin
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  if TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER then
    REPORT_ERROR;
  end if;
  case TOKEN.DELIMITER is
    when LEXICAL_ANALYZER.LESS_THAN | LEXICAL_ANALYZER.LESS_THAN_OR_EQUAL |
     LEXICAL_ANALYZER.GREATER_THAN | LEXICAL_ANALYZER.GREATER_THAN_OR_EQUAL
               => null;
    when others => REPORT_ERROR;
  end case;
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  PROCESS_RIGHT_COMPARISON_OPERAND
  ( TOKEN , SEMANTICALLY.BINARY_SQL_OPERATION ( TOKEN.DELIMITER ) , FROM ,
    T1 , R1 );
end PROCESS_INFIX_COMPARISON;

procedure VALIDATE_NOT_PROGRAM_BOOLEAN
          ( TOKEN                     : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            COMBINED_OPERAND_TYPE     : RESULT.DESCRIPTOR;
            OTHER_TYPE_INFORMATION : RESULT.DESCRIPTOR ) is
  COMBINED_TYPE : RESULT.DESCRIPTOR;
  COMPARABLE    : RESULT.COMPARABILITY;
  KNOWN_TYPE    : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
begin
  RESULT.COMBINED_TYPE
  ( COMBINED_OPERAND_TYPE , OTHER_TYPE_INFORMATION , COMBINED_TYPE ,
    COMPARABLE );
  if COMPARABLE = RESULT.IS_NOT_COMPARABLE then
    COMBINED_TYPE := COMBINED_OPERAND_TYPE;
  end if;
  KNOWN_TYPE := SEMANTICALLY.STRONGLY_TYPE ( COMBINED_TYPE );
  if KNOWN_TYPE /= null and then
   KNOWN_TYPE.ULT_PARENT_TYPE = PREDEFINED_TYPE.STANDARD.BOOLEAN and then
   COMBINED_OPERAND_TYPE.LOCATION = RESULT.IN_PROGRAM then
    LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
    ( TOKEN,
      "Use INDICATOR on one operand so both are not program BOOLEANs" );
  end if;
end VALIDATE_NOT_PROGRAM_BOOLEAN;

procedure PROCESS_BETWEEN_AND
          ( FROM             : in  FROM_CLAUSE.INFORMATION;
```

```
                LEFT_TYPE        : in  RESULT.DESCRIPTOR;
                RIGHT_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
                RIGHT_TYPE       : out RESULT.DESCRIPTOR ) is
  T1, T2      : TENTATIVE.FUNCTION_LIST;
  R1, R2, R3 : RESULT.DESCRIPTOR;
  COMPARABLE : RESULT.COMPARABILITY;
  TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN;
begin
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
  TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  SYNTACTICALLY.PROCESS_RESERVED_WORD ( LEXICAL_ANALYZER.R_AND );
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T2 , R2 );
  SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
  ( TOKEN , R1 , R2 , R3 , COMPARABLE );
  if COMPARABLE = RESULT.IS_COMPARABLE then
    VALIDATE_NOT_PROGRAM_BOOLEAN ( TOKEN , R3 , LEFT_TYPE );
  end if;
  R3.LOCATION := RESULT.IN_DATABASE;
  TENTATIVE.FUNCTION_REQUIRED_FOR_BINARY_OPERATION
  ( T1 , R3 , ADA_SQL_FUNCTION_DEFINITIONS.O_AND , R1 , R2 );
  RIGHT_FUNCTIONS := TENTATIVE.FUNCTION_LIST_MERGE ( T1 , T2 );
  RIGHT_TYPE := R3;
end PROCESS_BETWEEN_AND;

procedure PROCESS_BETWEEN_PREDICATE
          ( OPERATOR_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM           : FROM_CLAUSE.INFORMATION ) is
  T1, T2 : TENTATIVE.FUNCTION_LIST;
  R1, R2 : RESULT.DESCRIPTOR;
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.COMMA );
  PROCESS_BETWEEN_AND ( FROM , R1 , T2 , R2 );
  VALIDATE_COMPARABLE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
  ( OPERATOR_TOKEN , ADA_SQL_FUNCTION_DEFINITIONS.O_BETWEEN , T1 , T2 ,
    R1 , R2 , GENERATED_FUNCTIONS.O_SQL_OBJECT );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
end PROCESS_BETWEEN_PREDICATE;

procedure PROCESS_IN_VALUE_LIST
          ( FROM             : in  FROM_CLAUSE.INFORMATION;
            LEFT_TYPE        : in  RESULT.DESCRIPTOR;
            RIGHT_FUNCTIONS : out TENTATIVE.FUNCTION_LIST;
            RIGHT_TYPE       : out RESULT.DESCRIPTOR ) is
```

```
  TOKEN              : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  T1, T2             : TENTATIVE.FUNCTION_LIST;
  R1, R2, R3         : RESULT.DESCRIPTOR;
  COMPARABLE         : RESULT.COMPARABILITY;
  FIRST_OR           : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
  FIRST_COMPARABLE : RESULT.COMPARABILITY;
  FIRST_TYPE         : RESULT.DESCRIPTOR;
begin
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , TRUE , SEMANTICALLY.PROGRAM_VALUE , T1 , R1 );
  loop
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  exit when TOKEN.KIND /= LEXICAL_ANALYZER.RESERVED_WORD or else
   TOKEN.RESERVED_WORD /= LEXICAL_ANALYZER.R_OR;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    EXPRESSION.PROCESS_VALUE_EXPRESSION
    ( FROM , TRUE , SEMANTICALLY.PROGRAM_VALUE , T2 , R2 );
    SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
    ( TOKEN , R1 , R2 , R3 , COMPARABLE );
    if FIRST_OR = null then
      FIRST_OR := TOKEN;
      FIRST_COMPARABLE := COMPARABLE;
      FIRST_TYPE := R3;
    end if;
    T1 := TENTATIVE.FUNCTION_LIST_MERGE ( T1 , T2 );
    R3.LOCATION := RESULT.IN_DATABASE;
    TENTATIVE.FUNCTION_REQUIRED_FOR_BINARY_OPERATION
    ( T1 , R3 , ADA_SQL_FUNCTION_DEFINITIONS.O_OR , R1 , R2 );
    R1 := R3;
  end loop;
  if FIRST_OR /= null and then FIRST_COMPARABLE = RESULT.IS_COMPARABLE then
    VALIDATE_NOT_PROGRAM_BOOLEAN ( FIRST_OR , FIRST_TYPE , LEFT_TYPE );
  end if;
  RIGHT_FUNCTIONS := T1;
  RIGHT_TYPE := R1;
end PROCESS_IN_VALUE_LIST;

procedure PROCESS_IN_PREDICATE
          ( OPERATOR_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            KEYWORD        : PREDICATE_WORDS;
            FROM           : FROM_CLAUSE.INFORMATION ) is
  T1, T2          : TENTATIVE.FUNCTION_LIST;
  R1, R2          : RESULT.DESCRIPTOR;
  TOKEN           : LEXICAL_ANALYZER.LEXICAL_TOKEN;
  DOING_SUBQUERY : BOOLEAN;
  SELECT_TYPE     : SELEC.ROUTINE_NAME;
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
```

```
    EXPRESSION.PROCESS_VALUE_EXPRESSION
    ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T1 , R1 );
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.COMMA );
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    SEMANTICALLY.GET_SELECT_WORD ( TOKEN , DOING_SUBQUERY , SELECT_TYPE );
    if DOING_SUBQUERY then
      PROCESS_SUBQUERY ( SELECT_TYPE , FROM , T2 , R2 );
    else
      PROCESS_IN_VALUE_LIST ( FROM , R1 , T2 , R2 );
    end if;
    VALIDATE_COMPARABLE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
    ( OPERATOR_TOKEN , PREDICATE_OPERATION ( KEYWORD ) , T1 , T2 , R1 , R2 ,
      GENERATED_FUNCTIONS.O_SQL_OBJECT );
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  end PROCESS_IN_PREDICATE;

  function VALIDATE_PROGRAM_VALUE_USED ( R : RESULT.DESCRIPTOR )
   return BOOLEAN is
  begin
    if R.LOCATION = RESULT.IN_DATABASE then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN , "Program value required" );
      return FALSE;
    end if;
    return TRUE;
  end VALIDATE_PROGRAM_VALUE_USED;

  function VALIDATE_STRING ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
                             R     : RESULT.DESCRIPTOR ) return BOOLEAN is
    CLASS : DDL_DEFINITIONS.TYPE_TYPE;
  begin
    if R.TYPE_IS = RESULT.IS_UNKNOWN then
      CLASS := R.UNKNOWN_TYPE.CLASS;
    else
      CLASS := R.KNOWN_TYPE.WHICH_TYPE;
    end if;
    if CLASS /= DDL_DEFINITIONS.STR_ING then
      LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
      ( TOKEN , "String type required" );
      return FALSE;
    end if;
    return TRUE;
 end VALIDATE_STRING;

  procedure PROCESS_LIKE_PREDICATE ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
                                     FROM  : FROM_CLAUSE.INFORMATION ) is
    T1, T2     : TENTATIVE.FUNCTION_LIST;
    R1, R2, R3 : RESULT.DESCRIPTOR;
    COMPARABLE : RESULT.COMPARABILITY;
```

```
begin
  LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.LEFT_PARENTHESIS );
  EXPRESSION.PROCESS_COLUMN_SPECIFICATION ( FROM , FALSE , TRUE , T1 , R1 );
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.COMMA );
  EXPRESSION.PROCESS_VALUE_EXPRESSION
  ( FROM , FALSE , SEMANTICALLY.ANY_VALUE , T2 , R2 );
  if VALIDATE_PROGRAM_VALUE_USED ( R2 ) then
    SEMANTICALLY.VALIDATE_COMPARABLE_OPERANDS
    ( TOKEN , R1 , R2 , R3 , COMPARABLE );
    if COMPARABLE = RESULT.IS_COMPARABLE and then
     VALIDATE_STRING ( TOKEN , R3 ) then
      VALIDATE_AND_GENERATE_STRONGLY_TYPED_BINARY_OPERATION
      ( TOKEN , ADA_SQL_FUNCTION_DEFINITIONS.O_LIKE , T1 , T2 , R1 , R2 ,
        R3 , GENERATED_FUNCTIONS.O_SQL_OBJECT );
    end if;
  end if;
  SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
end PROCESS_LIKE_PREDICATE;

procedure PROCESS_PREDICATE
          ( TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            FROM  : FROM_CLAUSE.INFORMATION ) is
  PREDICATE_KEYWORD : PREDICATE_WORDS;
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.IDENTIFIER then
    begin
      PREDICATE_KEYWORD := PREDICATE_WORDS'VALUE ( TOKEN.ID.all );
    exception
      when CONSTRAINT_ERROR =>
        PROCESS_INFIX_COMPARISON ( FROM );
        return;
    end;
    case PREDICATE_KEYWORD is
      when EQ | NE         => PROCESS_PREFIX_COMPARISON
                              ( TOKEN , PREDICATE_KEYWORD , FROM );
      when BETWEEN         => PROCESS_BETWEEN_PREDICATE ( TOKEN , FROM );
      when IS_IN | NOT_IN => PROCESS_IN_PREDICATE
                              ( TOKEN , PREDICATE_KEYWORD , FROM );
      when LIKE            => PROCESS_LIKE_PREDICATE ( TOKEN , FROM );
    end case;
  else
    PROCESS_INFIX_COMPARISON ( FROM );
  end if;
end PROCESS_PREDICATE;

procedure PROCESS_BOOLEAN_PRIMARY ( FROM : FROM_CLAUSE.INFORMATION ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
```

```
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
   TOKEN.DELIMITER = LEXICAL_ANALYZER.LEFT_PARENTHESIS then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    PROCESS_SEARCH_CONDITION ( FROM );
    SYNTACTICALLY.PROCESS_DELIMITER ( LEXICAL_ANALYZER.RIGHT_PARENTHESIS );
  else
    PROCESS_PREDICATE ( TOKEN , FROM );
  end if;
end PROCESS_BOOLEAN_PRIMARY;

procedure PROCESS_BOOLEAN_FACTOR ( FROM : FROM_CLAUSE.INFORMATION ) is
  TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN :=
   LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
begin
  if TOKEN.KIND = LEXICAL_ANALYZER.RESERVED_WORD and then
   TOKEN.RESERVED_WORD = LEXICAL_ANALYZER.R_NOT then
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    GENERATED_FUNCTIONS.ADD_UNARY_FUNCTION
    ( OPERATION      => ADA_SQL_FUNCTION_DEFINITIONS.O_NOT,
      PARAMETER_KIND => GENERATED_FUNCTIONS.O_SQL_OBJECT,
      RESULT_KIND    => GENERATED_FUNCTIONS.O_SQL_OBJECT );
  end if;
  PROCESS_BOOLEAN_PRIMARY ( FROM );
end PROCESS_BOOLEAN_FACTOR;

procedure PROCESS_SEARCH_CONDITION ( FROM : FROM_CLAUSE.INFORMATION ) is
  OPERATOR_WAS_SEEN : BOOLEAN := FALSE;
  THE_OPERATOR_SEEN : LEXICAL_ANALYZER.RESERVED_WORD_KIND;
  TOKEN             : LEXICAL_ANALYZER.LEXICAL_TOKEN;
begin
  loop
    PROCESS_BOOLEAN_FACTOR ( FROM );
    TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
  exit when TOKEN.KIND /= LEXICAL_ANALYZER.RESERVED_WORD;
    case TOKEN.RESERVED_WORD is
      when R_AND | R_OR =>
        null;
      when others =>
        exit;
    end case;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    if not OPERATOR_WAS_SEEN then
      OPERATOR_WAS_SEEN := TRUE;
      THE_OPERATOR_SEEN := TOKEN.RESERVED_WORD;
      GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
      ( OPERATION            => BOOLEAN_OPERATION ( THE_OPERATOR_SEEN ),
        LEFT_PARAMETER_KIND  => GENERATED_FUNCTIONS.O_SQL_OBJECT,
        RIGHT_PARAMETER_KIND => GENERATED_FUNCTIONS.O_SQL_OBJECT,
```

```
          RESULT_KIND          => GENERATED_FUNCTIONS.O_SQL_OBJECT );
      else
        if THE_OPERATOR_SEEN /= TOKEN.RESERVED_WORD then
          LEXICAL_ANALYZER.REPORT_SEMANTIC_ERROR
          ( TOKEN , "Mixed ANDs and ORs must be parenthesized" );
        end if;
      end if;
    end loop;
  end PROCESS_SEARCH_CONDITION;

end SEARCH_CONDITION;
```

3.11.83 package tblexprb.ada

```
with LEXICAL_ANALYZER, FROM_CLAUSE, DDL_DEFINITIONS, TABLE, CORRELATION,
     UNQUALIFIED_NAME, GENERATED_FUNCTIONS, SEARCH_CONDITION, EXPRESSION,
     TENTATIVE, RESULT, ADA_SQL_FUNCTION_DEFINITIONS;
use  LEXICAL_ANALYZER, DDL_DEFINITIONS, CORRELATION;

package body TABLE_EXPRESSION is

---------------------------------------------------------------------------
-- GOT_FROM_AMPERSAND - read token and gobble it and return true if it's &
--                      otherwise return false

  function GOT_FROM_AMPERSAND
           return BOOLEAN is

    AMPERSAND_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    AMPERSAND_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if AMPERSAND_TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
       AMPERSAND_TOKEN.DELIMITER = LEXICAL_ANALYZER.AMPERSAND then
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      return TRUE;
    else
      return FALSE;
    end if;
  end GOT_FROM_AMPERSAND;

-----------------------------------------------------------------------------
-- PROCESS_TABLE_REFERENCE -

  procedure PROCESS_TABLE_REFERENCE
           (SCOPE                : FROM_CLAUSE.INFORMATION;
            RETURNS_TABLE_LIST : BOOLEAN;
            TABLE_TOKEN          : LEXICAL_ANALYZER.LEXICAL_TOKEN;
            CORRELATION_TOKEN    : LEXICAL_ANALYZER.LEXICAL_TOKEN) is
```

```
  TABLE_DES              : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
  STATUS                 : CORRELATION.NAME_REFERENCE_STATUS;
  CORRELATION_NAME       : CORRELATION.NAME_DECLARED_ENTRY;
  DUMMY_TABLE            : DDL_DEFINITIONS.ACCESS_TYPE_DESCRIPTOR;
  DUMMY_CORRELATION_NAME : CORRELATION.NAME_DECLARED_ENTRY;
  TABLE_STATUS           : TABLE.NAME_STATUS;

begin
  TABLE.DESCRIPTOR_FOR (TABLE_TOKEN.ID.all, TABLE_STATUS, TABLE_DES);
  case TABLE_STATUS is
    when TABLE.NAME_UNDEFINED =>
               LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
               "Table name is undefined");
    when TABLE.NAME_AMBIGUOUS =>
               LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
               "Table name is ambiguous");
    when TABLE.NAME_UNIQUE    => null;
  end case;
  if CORRELATION_TOKEN = null then
    FROM_CLAUSE.EXPOSES_NAME (TABLE_TOKEN.ID.all, SCOPE, TRUE, DUMMY_TABLE,
                              DUMMY_CORRELATION_NAME);
    if DUMMY_TABLE /= null or else DUMMY_CORRELATION_NAME /= null then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
      "Table name already used in from clause");
    end if;
    FROM_CLAUSE.NAMES_EXPOSED_TABLE (SCOPE, TABLE_DES);
    if RETURNS_TABLE_LIST then
      UNQUALIFIED_NAME.RETURNS_TABLE_LIST (TABLE_DES.FULL_NAME.NAME);
    else
      UNQUALIFIED_NAME.RETURNS_TABLE_NAME (TABLE_DES.FULL_NAME.NAME);
    end if;
  else
    if RETURNS_TABLE_LIST then
      CORRELATION.NAME_RETURNS_TABLE_LIST (CORRELATION_TOKEN.ID.all,
                  TABLE_DES, STATUS, CORRELATION_NAME);
    else
      CORRELATION.NAME_RETURNS_TABLE_NAME (CORRELATION_TOKEN.ID.all,
                  TABLE_DES, STATUS, CORRELATION_NAME);
    end if;
    case STATUS is
      when CORRELATION.NAME_VALID => null;
      when CORRELATION.NAME_NOT_DECLARED =>
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
          "Correlation name has not been declared");
      when CORRELATION.NAME_DECLARED_FOR_DIFFERENT_TABLE =>
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
          "Correlation name has already been declared for another table");
    end case;
    FROM_CLAUSE.EXPOSES_NAME (CORRELATION_TOKEN.ID.all, SCOPE, TRUE,
```

```
                 DUMMY_TABLE, DUMMY_CORRELATION_NAME);
      if DUMMY_TABLE /= null or else DUMMY_CORRELATION_NAME /= null then
        LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
        "Correlation name has already been declared for another table");
      end if;
      FROM_CLAUSE.NAMES_CORRELATED_TABLE (SCOPE, CORRELATION_NAME);
    end if;
  end PROCESS_TABLE_REFERENCE;

--------------------------------------------------------------------
-- GOT_FROM_TABLE - reads tokens for a table or correlation.table and
--                  processes them accordingly.  Return true after one
--                  is successfully processed.

  function GOT_FROM_TABLE
          (SCOPE       : FROM_CLAUSE.INFORMATION;
           FIRST_TABLE : BOOLEAN)
           return        BOOLEAN is

    TABLE_TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    DOT_TOKEN         : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;
    CORRELATION_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN := null;

  begin
    CORRELATION_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if CORRELATION_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (CORRELATION_TOKEN,
      "Expecting table name");
    else
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      DOT_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if DOT_TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
         DOT_TOKEN.DELIMITER = LEXICAL_ANALYZER.DOT then
        TABLE_TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
        if TABLE_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER then
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (TABLE_TOKEN,
          "Expecting correlation_name.table_name");
        end if;
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
        LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      else
        TABLE_TOKEN := CORRELATION_TOKEN;
        CORRELATION_TOKEN := null;
        DOT_TOKEN := null;
      end if;
    end if;
    PROCESS_TABLE_REFERENCE (SCOPE, FIRST_TABLE, TABLE_TOKEN,
                             CORRELATION_TOKEN);
    return TRUE;
```

```
  end GOT_FROM_TABLE;

----------------------------------------------------------------------------
-- GOT_FROM_CLAUSE - we should now find FROM =>  tokens.  If not print
--                   error message.  If we do return true

  function GOT_FROM_CLAUSE
           return BOOLEAN is

    FROM_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;

  begin
    FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    case FROM_TOKEN.KIND is
      when LEXICAL_ANALYZER.IDENTIFIER        =>
        if FROM_TOKEN.ID.all = "FROM" then
          LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
          FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
          case FROM_TOKEN.KIND is
            when LEXICAL_ANALYZER.DELIMITER  =>
                 if FROM_TOKEN.DELIMITER = LEXICAL_ANALYZER.ARROW then
                   LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
                   return TRUE;
                 end if;
            when others    =>  null;
          end case;
          LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (FROM_TOKEN,
          "Expecting token: =>");
        end if;
      when others              => null;
    end case;
    LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (FROM_TOKEN,
    "Expecting token: FROM");
  end GOT_FROM_CLAUSE;

----------------------------------------------------------------------------
-- PROCESS_FROM_CLAUSE - process a from clause

  procedure PROCESS_FROM_CLAUSE
           (SCOPE : FROM_CLAUSE.INFORMATION) is

    FIRST_TABLE     : BOOLEAN := TRUE;
    DONE_AMPERSAND  : BOOLEAN := FALSE;

  begin
    if GOT_FROM_CLAUSE then
      loop
        exit when not GOT_FROM_TABLE (SCOPE, FIRST_TABLE);
        FIRST_TABLE := FALSE;
```

```
        exit when not GOT_FROM_AMPERSAND;
        if not DONE_AMPERSAND then
          DONE_AMPERSAND := TRUE;
          GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
                (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND,
                 GENERATED_FUNCTIONS.O_TABLE_LIST, null,
                 GENERATED_FUNCTIONS.O_TABLE_NAME, null,
                 GENERATED_FUNCTIONS.O_TABLE_LIST, null);
        end if;
      end loop;
    end if;
  end PROCESS_FROM_CLAUSE;

----------------------------------------------------------------------------
-- SKIP_OVER_FROM_CLAUSE - we should now find FROM and skip over the from
--                         clause ending on a , or )

  procedure SKIP_OVER_FROM_CLAUSE is

    FROM_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;

  begin
    FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if FROM_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER or else
       FROM_TOKEN.ID.all /= "FROM" then
      return;
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    loop
    FROM_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    exit when FROM_TOKEN.KIND = LEXICAL_ANALYZER.END_OF_FILE;
    exit when FROM_TOKEN.KIND = LEXICAL_ANALYZER.DELIMITER and then
             (FROM_TOKEN.DELIMITER = LEXICAL_ANALYZER.COMMA or
              FROM_TOKEN.DELIMITER = LEXICAL_ANALYZER.RIGHT_PARENTHESIS);
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    end loop;
  end SKIP_OVER_FROM_CLAUSE;

----------------------------------------------------------------------
-- PROCESS_WHERE_CLAUSE

  procedure PROCESS_WHERE_CLAUSE
           (FROM_INFO : FROM_CLAUSE.INFORMATION) is

    WHERE_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;

  begin
    WHERE_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if WHERE_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
```

```
        WHERE_TOKEN.DELIMITER /= LEXICAL_ANALYZER.COMMA then
      return;
    end if;
    WHERE_TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
    if WHERE_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER or else
        WHERE_TOKEN.ID.all /= "WHERE" then
      return;
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    WHERE_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if WHERE_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
        WHERE_TOKEN.DELIMITER /= LEXICAL_ANALYZER.ARROW then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (WHERE_TOKEN,
      "Expecting => in WHERE clause");
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    SEARCH_CONDITION.PROCESS_SEARCH_CONDITION (FROM_INFO);
  end PROCESS_WHERE_CLAUSE;

----------------------------------------------------------------------
-- PROCESS_GROUP_BY_CLAUSE

  procedure PROCESS_GROUP_BY_CLAUSE
           (FROM_INFO : FROM_CLAUSE.INFORMATION) is

    GROUP_TOKEN       : LEXICAL_ANALYZER.LEXICAL_TOKEN;
    NEEDED_FUNCTIONS  : TENTATIVE.FUNCTION_LIST;
    RESULTS           : RESULT.DESCRIPTOR;
    DONE_AMPERSAND    : BOOLEAN := FALSE;

  begin
    GROUP_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if GROUP_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
        GROUP_TOKEN.DELIMITER /= LEXICAL_ANALYZER.COMMA then
      return;
    end if;
    GROUP_TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
    if GROUP_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER or else
        GROUP_TOKEN.ID.all /= "GROUP_BY" then
      return;
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    GROUP_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if GROUP_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
        GROUP_TOKEN.DELIMITER /= LEXICAL_ANALYZER.ARROW then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (GROUP_TOKEN,
      "Expecting => in GROUP_BY clause");
```

```
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    loop
      EXPRESSION.PROCESS_COLUMN_SPECIFICATION (FROM_INFO, TRUE, FALSE,
                    NEEDED_FUNCTIONS, RESULTS);
      TENTATIVE.FUNCTIONS_RETURN_SQL_OBJECT (NEEDED_FUNCTIONS);
      GROUP_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
      if GROUP_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
         GROUP_TOKEN.DELIMITER /= LEXICAL_ANALYZER.AMPERSAND then
        exit;
      end if;
      LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
      if not DONE_AMPERSAND then
        DONE_AMPERSAND := TRUE;
        GENERATED_FUNCTIONS.ADD_BINARY_FUNCTION
            (ADA_SQL_FUNCTION_DEFINITIONS.O_AMPERSAND,
             GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
             GENERATED_FUNCTIONS.O_SQL_OBJECT, null,
             GENERATED_FUNCTIONS.O_SQL_OBJECT, null);
      end if;
    end loop;
  end PROCESS_GROUP_BY_CLAUSE;

----------------------------------------------------------------------
-- PROCESS_HAVING_CLAUSE_CLAUSE

  procedure PROCESS_HAVING_CLAUSE_CLAUSE
           (FROM_INFO : FROM_CLAUSE.INFORMATION) is

    HAVING_TOKEN : LEXICAL_ANALYZER.LEXICAL_TOKEN;

  begin
    HAVING_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if HAVING_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
       HAVING_TOKEN.DELIMITER /= LEXICAL_ANALYZER.COMMA then
      return;
    end if;
    HAVING_TOKEN := LEXICAL_ANALYZER.NEXT_LOOK_AHEAD_TOKEN;
    if HAVING_TOKEN.KIND /= LEXICAL_ANALYZER.IDENTIFIER or else
       HAVING_TOKEN.ID.all /= "HAVING" then
      return;
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    HAVING_TOKEN := LEXICAL_ANALYZER.FIRST_LOOK_AHEAD_TOKEN;
    if HAVING_TOKEN.KIND /= LEXICAL_ANALYZER.DELIMITER or else
       HAVING_TOKEN.DELIMITER /= LEXICAL_ANALYZER.ARROW then
      LEXICAL_ANALYZER.REPORT_SYNTAX_ERROR (HAVING_TOKEN,
      "Expecting => in HAVING clause");
```

```
    end if;
    LEXICAL_ANALYZER.EAT_NEXT_TOKEN;
    SEARCH_CONDITION.PROCESS_SEARCH_CONDITION (FROM_INFO);
  end PROCESS_HAVING_CLAUSE_CLAUSE;

----------------------------------------------------------------------------
-- PROCESS_REST_OF_TABLE_EXPRESSION

  procedure PROCESS_REST_OF_TABLE_EXPRESSION
           (SCOPE : FROM_CLAUSE.INFORMATION) is
  begin
    SKIP_OVER_FROM_CLAUSE;
    PROCESS_WHERE_CLAUSE (SCOPE);
    PROCESS_GROUP_BY_CLAUSE (SCOPE);
    PROCESS_HAVING_CLAUSE_CLAUSE (SCOPE);
  end PROCESS_REST_OF_TABLE_EXPRESSION;
end TABLE_EXPRESSION;
```

3.11.84 package ddl_schema_io_internal_spec.ada

```
with TEXT_IO, IO_DEFINITIONS, DDL_DEFINITIONS, IO_ERRORS;
use  TEXT_IO, IO_DEFINITIONS, DDL_DEFINITIONS, IO_ERRORS;

package IO_INTERNAL_STUFF is

  procedure TOKEN_END          -- internal, find end of token
           (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            T_END  : out POSITIVE);

  function WHITESPACE          -- internal, is character white space
          (C : in CHARACTER)
           return BOOLEAN;

  function ALPHABETIC          -- internal, is character alphabetic
          (C : in CHARACTER)
           return BOOLEAN;

  function SIMPLE_NUMERIC      -- internal, is character simple numeric
          (C : in CHARACTER)
           return BOOLEAN;

  function QUALIFIER           -- internal, is character a qualifier
          (C     : in CHARACTER;
           BUF   : in STRING;
           PTR   : in NATURAL;
           FIRST : in POSITIVE;
           LAST  : in NATURAL)
           return BOOLEAN;

  procedure NUMERIC
```

```
            (OK    : out BOOLEAN;
             C     : in CHARACTER;
             DOT   : in out BOOLEAN;
             EXP   : in out NATURAL;
             PTR   : in NATURAL;
             FIRST : in POSITIVE;
             LAST  : in POSITIVE;
             BUF   : in STRING);

  function VALID_AFTER_DECIMAL
            (C : in CHARACTER)
             return BOOLEAN;

  procedure NEXT_TOKEN         -- internal, set up to point to next token
             (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure NEXT_LINE          -- internal, read next schema unit line
             (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR);

end IO_INTERNAL_STUFF;
```

3.11.85 package ddl_schema_io_spec.ada

```
with TEXT_IO, DATABASE, IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS,
     IO_INTERNAL_STUFF, IO_ERRORS, LEXICAL_ANALYZER;
use  TEXT_IO, DATABASE, IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS,
     IO_INTERNAL_STUFF, IO_ERRORS;

package SCHEMA_IO is

  procedure OPEN_SCHEMA_UNIT   -- open an Ada schema unit file
             (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure GET_STRING         -- get next token into temp_string
             (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
              STR     : out STRING;
              LAST    : out NATURAL);

  procedure CLOSE_SCHEMA_UNIT  -- close the schema unit
             (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure PRINT_ERROR        -- print error to file, increment fatal_error
             (MESSAGE : in STRING);

  procedure PRINT_TO_FILE      -- print message to output file
             (MESSAGE : in STRING);

  procedure PRINT_MESSAGE      -- print message on terminal
             (MESSAGE : in STRING);
```

```
  procedure GET_TERMINAL_INPUT -- read input from terminal
          (MESSAGE : in out STRING;
           LENGTH  : in out NATURAL);

  procedure OPEN_OUTPUT_FILE   -- open the output disk file
          (NAME : in STRING);

  procedure CLOSE_OUTPUT_FILE; -- close the output disk file

  procedure UPPER_CASE         -- convert string to upper case
          (LINE : in out STRING);

  procedure LOWER_CASE         -- convert string to lower case
          (LINE : in out STRING);

  function DOUBLE_PRECISION_TO_STRING     -- convert double precision to string
         (NUM : in DOUBLE_PRECISION)
          return STRING;

  procedure STRING_TO_DOUBLE_PRECISION  -- convert string to double_precision
         (NUM_STRING   : in STRING;
          OK           : in out BOOLEAN;
          NUM          : out DOUBLE_PRECISION);

  procedure EXCHANGE_FOR_ORIGINAL
          (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           BUF     : in out STRING;
           BUF_LEN : in out NATURAL);

  procedure GET_SINGLE_QUOTE_STRING
          (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           BUF     : in out STRING;
           BUF_LEN : in out NATURAL;
           VALID   : out BOOLEAN);

end SCHEMA_IO;
```

3.11.86 package ddl_new_des.ada

```
package body GET_NEW_DESCRIPTOR_ROUTINES is

----------------------------------------------------------------------------
--
-- GET_NEW_YET_TO_DO_DESCRIPTOR
--

  function GET_NEW_YET_TO_DO_DESCRIPTOR
          return ACCESS_YET_TO_DO_DESCRIPTOR is
  begin
    return new YET_TO_DO_DESCRIPTOR'
```

```
        (UNDONE_SCHEMA            => null,
         PREVIOUS_YET_TO_DO       => null,
         NEXT_YET_TO_DO           => null);
  end GET_NEW_YET_TO_DO_DESCRIPTOR;

--------------------------------------------------------------------------------
--
-- GET_NEW_SCHEMA_UNIT_DESCRIPTOR
--

  function GET_NEW_SCHEMA_UNIT_DESCRIPTOR
           return ACCESS_SCHEMA_UNIT_DESCRIPTOR is
  begin
    return new SCHEMA_UNIT_DESCRIPTOR'
        (NAME                      => null,
         AUTH_ID                   => null,
         IS_AUTH_PACKAGE           => FALSE,
         HAS_DECLARED_TYPES        => FALSE,
         HAS_DECLARED_TABLES       => FALSE,
         HAS_DECLARED_VARIABLES    => FALSE,
         FIRST_WITHED              => null,
         LAST_WITHED               => null,
         FIRST_USED                => null,
         LAST_USED                 => null,
         FIRST_DECLARED_PACKAGE    => null,
         LAST_DECLARED_PACKAGE     => null,
         STREAM                    => null,
         SCHEMA_STATUS             => NOTOPEN,
         PREVIOUS_SCHEMA_UNIT      => null,
         NEXT_SCHEMA_UNIT          => null);
  end GET_NEW_SCHEMA_UNIT_DESCRIPTOR;

--------------------------------------------------------------------------------
--
-- GET_NEW_WITHED_UNIT_DESCRIPTOR
--

  function GET_NEW_WITHED_UNIT_DESCRIPTOR
           return ACCESS_WITHED_UNIT_DESCRIPTOR is
  begin
    return new WITHED_UNIT_DESCRIPTOR'
        (SCHEMA_UNIT       => null,
         PREVIOUS_WITHED   => null,
         NEXT_WITHED       => null);
  end GET_NEW_WITHED_UNIT_DESCRIPTOR;

--------------------------------------------------------------------------------
--
-- GET_NEW_USED_PACKAGE_DESCRIPTOR
```

```
--

  function GET_NEW_USED_PACKAGE_DESCRIPTOR
           return ACCESS_USED_PACKAGE_DESCRIPTOR is
  begin
    return new USED_PACKAGE_DESCRIPTOR'
        (NAME              => null,
         PREVIOUS_USED     => null,
         NEXT_USED         => null);
  end GET_NEW_USED_PACKAGE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_DECLARED_PACKAGE_DESCRIPTOR
--

  function GET_NEW_DECLARED_PACKAGE_DESCRIPTOR
           return ACCESS_DECLARED_PACKAGE_DESCRIPTOR is
  begin
    return new DECLARED_PACKAGE_DESCRIPTOR'
        (NAME                 => null,
         FOUND_END            => FALSE,
         PREVIOUS_DECLARED    => null,
         NEXT_DECLARED        => null);
  end GET_NEW_DECLARED_PACKAGE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_IDENTIFIER_DESCRIPTOR
--

  function GET_NEW_IDENTIFIER_DESCRIPTOR
           return ACCESS_IDENTIFIER_DESCRIPTOR is
  begin
    return new IDENTIFIER_DESCRIPTOR'
        (NAME             => null,
         FIRST_FULL_NAME  => null,
         LAST_FULL_NAME   => null,
         PREVIOUS_IDENT   => null,
         NEXT_IDENT       => null);
  end GET_NEW_IDENTIFIER_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_FULL_NAME_DESCRIPTOR
--

  function GET_NEW_FULL_NAME_DESCRIPTOR
           return ACCESS_FULL_NAME_DESCRIPTOR is
```

```
  begin
    return new FULL_NAME_DESCRIPTOR'
       (NAME                  => null,
        FULL_PACKAGE_NAME     => null,
        TABLE_NAME            => null,
        IS_NOT_NULL           => FALSE,
        IS_NOT_NULL_UNIQUE    => FALSE,
        TYPE_IS               => null,
        SCHEMA_UNIT           => null,
        PREVIOUS_NAME         => null,
        NEXT_NAME             => null);
  end GET_NEW_FULL_NAME_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR FOR RECORD
--

  function GET_NEW_RECORD_DESCRIPTOR
           return ACCESS_RECORD_DESCRIPTOR is
  begin
    return new TYPE_DESCRIPTOR'
       (TY_PE              => REC_ORD,
        TYPE_KIND          => A_TYPE,
        WHICH_TYPE         => REC_ORD,
        FULL_NAME          => null,
        NOT_NULL           => FALSE,
        NOT_NULL_UNIQUE    => FALSE,
        FIRST_SUBTYPE      => null,
        LAST_SUBTYPE       => null,
        FIRST_DERIVED      => null,
        LAST_DERIVED       => null,
        FIRST_COMPONENT    => null,
        LAST_COMPONENT     => null,
        PREVIOUS_ONE       => null,
        NEXT_ONE           => null,
        PREVIOUS_TYPE      => null,
        NEXT_TYPE          => null,
        ULT_PARENT_TYPE    => null,
        PARENT_TYPE        => null,
        BASE_TYPE          => null,
        PARENT_RECORD      => null);
  end GET_NEW_RECORD_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR FOR ENUMERATION
--
```

```
  function GET_NEW_ENUMERATION_DESCRIPTOR
           return ACCESS_ENUMERATION_DESCRIPTOR is
  begin
    return new TYPE_DESCRIPTOR'
       (TY_PE              => ENUMERATION,
        TYPE_KIND          => A_TYPE,
        WHICH_TYPE         => ENUMERATION,
        FULL_NAME          => null,
        NOT_NULL           => FALSE,
        NOT_NULL_UNIQUE    => FALSE,
        FIRST_SUBTYPE      => null,
        LAST_SUBTYPE       => null,
        FIRST_DERIVED      => null,
        LAST_DERIVED       => null,
        FIRST_COMPONENT    => null,
        LAST_COMPONENT     => null,
        PREVIOUS_ONE       => null,
        NEXT_ONE           => null,
        PREVIOUS_TYPE      => null,
        NEXT_TYPE          => null,
        ULT_PARENT_TYPE    => null,
        PARENT_TYPE        => null,
        BASE_TYPE          => null,
        PARENT_RECORD      => null,
        FIRST_LITERAL      => null,
        LAST_LITERAL       => null,
        LAST_POS           => 0,
        MAX_LENGTH         => 0);
  end GET_NEW_ENUMERATION_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR FOR INTEGER
--

  function GET_NEW_INTEGER_DESCRIPTOR
           return ACCESS_INTEGER_DESCRIPTOR is
  begin
    return new TYPE_DESCRIPTOR'
       (TY_PE              => INT_EGER,
        TYPE_KIND          => A_TYPE,
        WHICH_TYPE         => INT_EGER,
        FULL_NAME          => null,
        NOT_NULL           => FALSE,
        NOT_NULL_UNIQUE    => FALSE,
        FIRST_SUBTYPE      => null,
        LAST_SUBTYPE       => null,
        FIRST_DERIVED      => null,
        LAST_DERIVED       => null,
```

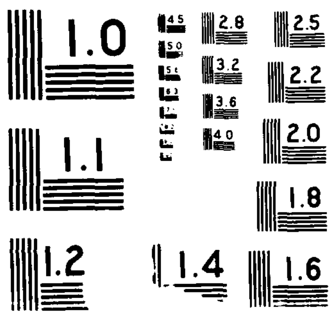
```
         FIRST_COMPONENT  => null,
         LAST_COMPONENT   => null,
         PREVIOUS_ONE     => null,
         NEXT_ONE         => null,
         PREVIOUS_TYPE    => null,
         NEXT_TYPE        => null,
         ULT_PARENT_TYPE  => null,
         PARENT_TYPE      => null,
         BASE_TYPE        => null,
         PARENT_RECORD    => null,
         RANGE_LO_INT     => -1,
         RANGE_HI_INT     => -1);
  end GET_NEW_INTEGER_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR FOR FLOAT
--

  function GET_NEW_FLOAT_DESCRIPTOR
           return ACCESS_FLOAT_DESCRIPTOR is
  begin
    return new TYPE_DESCRIPTOR'
        (TY_PE            => FL_OAT,
         TYPE_KIND        => A_TYPE,
         WHICH_TYPE       => FL_OAT,
         FULL_NAME        => null,
         NOT_NULL         => FALSE,
         NOT_NULL_UNIQUE  => FALSE,
         FIRST_SUBTYPE    => null,
         LAST_SUBTYPE     => null,
         FIRST_DERIVED    => null,
         LAST_DERIVED     => null,
         FIRST_COMPONENT  => null,
         LAST_COMPONENT   => null,
         PREVIOUS_ONE     => null,
         NEXT_ONE         => null,
         PREVIOUS_TYPE    => null,
         NEXT_TYPE        => null,
         ULT_PARENT_TYPE  => null,
         PARENT_TYPE      => null,
         BASE_TYPE        => null,
         PARENT_RECORD    => null,
         FLOAT_DIGITS     => 0,
         RANGE_LO_FLT     => -1.0,
         RANGE_HI_FLT     => -1.0);
  end GET_NEW_FLOAT_DESCRIPTOR;

----------------------------------------------------------------------------
```

```
--
-- GET_NEW_TYPE_DESCRIPTOR FOR STRING
--

  function GET_NEW_STRING_DESCRIPTOR
           return ACCESS_STRING_DESCRIPTOR is
  begin
    return new TYPE_DESCRIPTOR'
       (TY_PE              => STR_ING,
        TYPE_KIND          => A_TYPE,
        WHICH_TYPE         => STR_ING,
        FULL_NAME          => null,
        NOT_NULL           => FALSE,
        NOT_NULL_UNIQUE    => FALSE,
        FIRST_SUBTYPE      => null,
        LAST_SUBTYPE       => null,
        FIRST_DERIVED      => null,
        LAST_DERIVED       => null,
        FIRST_COMPONENT    => null,
        LAST_COMPONENT     => null,
        PREVIOUS_ONE       => null,
        NEXT_ONE           => null,
        PREVIOUS_TYPE      => null,
        NEXT_TYPE          => null,
        ULT_PARENT_TYPE    => null,
        PARENT_TYPE        => null,
        BASE_TYPE          => null,
        PARENT_RECORD      => null,
        LENGTH             => 0,
        INDEX_TYPE         => null,
        ARRAY_TYPE         => null,
        CONSTRAINED        => FALSE,
        ARRAY_RANGE_LO     => -1,
        ARRAY_RANGE_HI     => -1,
        ARRAY_RANGE_MAX    => -1,
        ARRAY_RANGE_MIN    => -1);
  end GET_NEW_STRING_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR FOR RECORD, ENUMERATION, INTEGER, FLOAT or STRING
--

  function GET_NEW_TYPE_DESCRIPTOR
          (IN_TYPE     : in TYPE_TYPE)
           return ACCESS_TYPE_DESCRIPTOR is
  begin
    case IN_TYPE is
      when REC_ORD       => return GET_NEW_RECORD_DESCRIPTOR;
```

```
      when ENUMERATION => return GET_NEW_ENUMERATION_DESCRIPTOR;
      when INT_EGER    => return GET_NEW_INTEGER_DESCRIPTOR;
      when FL_OAT      => return GET_NEW_FLOAT_DESCRIPTOR;
      when STR_ING     => return GET_NEW_STRING_DESCRIPTOR;
    end case;
  end GET_NEW_TYPE_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_LITERAL_DESCRIPTOR
--

  function GET_NEW_LITERAL_DESCRIPTOR
      return ACCESS_LITERAL_DESCRIPTOR is
  begin
    return new LITERAL_DESCRIPTOR'
       (NAME               => null,
        POS                => 0,
        PARENT_ENUM        => null,
        PREVIOUS_LITERAL   => null,
        NEXT_LITERAL       => null);
  end GET_NEW_LITERAL_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_ENUM_LIT_DESCRIPTOR
--

  function GET_NEW_ENUM_LIT_DESCRIPTOR
      return ACCESS_ENUM_LIT_DESCRIPTOR is
  begin
    return new ENUM_LIT_DESCRIPTOR'
       (NAME                 => null,
        FIRST_FULL_ENUM_LIT  => null,
        LAST_FULL_ENUM_LIT   => null,
        PREVIOUS_ENUM_LIT    => null,
        NEXT_ENUM_LIT        => null);
  end GET_NEW_ENUM_LIT_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- GET_NEW_FULL_ENUM_LIT_DESCRIPTOR
--

  function GET_NEW_FULL_ENUM_LIT_DESCRIPTOR
      return ACCESS_FULL_ENUM_LIT_DESCRIPTOR is
  begin
    return new FULL_ENUM_LIT_DESCRIPTOR'
       (NAME           => null,
```

```
      TYPE_IS         => null,
      PREVIOUS_LIT    => null,
      NEXT_LIT        => null);
  end GET_NEW_FULL_ENUM_LIT_DESCRIPTOR;

-----------------------------------------------------------------------------
--
-- GET_NEW_ENUM_LIT_NAME
--

  function GET_NEW_ENUM_LIT_NAME
          (TEMP : in STRING)
           return ENUM_LIT_NAME is
  begin
    return new ENUM_LIT_NAME_STRING' (ENUM_LIT_NAME_STRING (TEMP));
  end GET_NEW_ENUM_LIT_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_AUTH_IDENT_NAME
--

  function GET_NEW_AUTH_IDENT_NAME
          (TEMP : in STRING)
           return AUTH_IDENT_NAME is
  begin
    return new AUTH_IDENT_NAME_STRING' (AUTH_IDENT_NAME_STRING (TEMP));
  end GET_NEW_AUTH_IDENT_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_LIBRARY_UNIT_NAME
--

  function GET_NEW_LIBRARY_UNIT_NAME
          (TEMP : in STRING)
           return LIBRARY_UNIT_NAME is
  begin
    return new LIBRARY_UNIT_NAME_STRING' (LIBRARY_UNIT_NAME_STRING (TEMP));
  end GET_NEW_LIBRARY_UNIT_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_PACKAGE_NAME
--

  function GET_NEW_PACKAGE_NAME
          (TEMP : in STRING)
           return PACKAGE_NAME is
```

AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION 5/6
SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA
B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317
UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

```
  begin
    return new PACKAGE_NAME_STRING' (PACKAGE_NAME_STRING (TEMP));
  end GET_NEW_PACKAGE_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_RECORD_NAME
--

  function GET_NEW_RECORD_NAME
          (TEMP : in STRING)
           return RECORD_NAME is
  begin
    return new RECORD_NAME_STRING' (RECORD_NAME_STRING (TEMP));
  end GET_NEW_RECORD_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_TYPE_NAME
--

  function GET_NEW_TYPE_NAME
          (TEMP : in STRING)
           return TYPE_NAME is
  begin
    return new TYPE_NAME_STRING' (TYPE_NAME_STRING (TEMP));
  end GET_NEW_TYPE_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_ENUMERATION_NAME
--

  function GET_NEW_ENUMERATION_NAME
          (TEMP : in STRING)
           return ENUMERATION_NAME is
  begin
    return new ENUMERATION_NAME_STRING' (ENUMERATION_NAME_STRING (TEMP));
  end GET_NEW_ENUMERATION_NAME;

end GET_NEW_DESCRIPTOR_ROUTINES;
```

3.11.87 package ddl_schema_io.ada

```
package body SCHEMA_IO is

--------------------------------------------------------------------------
--
-- OPEN_SCHEMA_UNIT
--
```

```
-- if the file is not and has not been processed then set the file name up to
-- be the library unit plus the extention of .ADA or .A or what ever is
-- defined in ddl_io_defs as being the extention of the system.  Open the file
-- and set the status to processing.  If we get an exception on opening the
-- file print the appropriate message and set status to not found.

  procedure OPEN_SCHEMA_UNIT
          (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is

  ADA_NAME   : STRING (1..250) := (others => ' ');
  LENGTH     : NATURAL := 0;

  begin
    if SCHEMA.SCHEMA_STATUS = NOTOPEN then
      if STRING (SCHEMA.NAME.all) = STANDARD_NAME then
        LENGTH := STANDARD_NAME_FILE'LAST;
        ADA_NAME (1..LENGTH) := STANDARD_NAME_FILE;
      elsif STRING (SCHEMA.NAME.all) = DATABASE_NAME then
        LENGTH := DATABASE_NAME_FILE'LAST;
        ADA_NAME (1..LENGTH) := DATABASE_NAME_FILE;
      elsif STRING (SCHEMA.NAME.all) = CURSOR_NAME then
        LENGTH := CURSOR_NAME_FILE'LAST;
        ADA_NAME (1..LENGTH) := CURSOR_NAME_FILE;
      else
        LENGTH := SCHEMA.NAME'LAST;
        ADA_NAME (1..LENGTH) := STRING(SCHEMA.NAME.all);
      end if;
      if ADA_NAME (1) in 'a'..'z' then
      LOWER_CASE (DOT_ADA_DEFAULT);
      else
      UPPER_CASE (DOT_ADA_DEFAULT);
      end if;
      ADA_NAME (LENGTH+1..LENGTH+DOT_ADA_LEN) := DOT_ADA_DEFAULT;
      LENGTH := LENGTH + DOT_ADA_LEN;
      SCHEMA.STREAM := new INPUT_RECORD;
      if DEBUGGING then
        PRINT_TO_FILE ("*** Opening schema unit: " & ADA_NAME (1..LENGTH));
      end if;
      if WHERE_IS_SCHEMA_FROM = CALLS and STRING(SCHEMA.NAME.all) =
              SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN) then
        null;
      else
        OPEN (SCHEMA.STREAM.FILE, IN_FILE, ADA_NAME (1..LENGTH));
      end if;
      SCHEMA.SCHEMA_STATUS := PROCESSING;
    end if;

  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
```

```
      OPEN_ERROR (SCHEMA, "Status", ADA_NAME (1..LENGTH));
    when MODE_ERROR =>      -- read output or write input
      OPEN_ERROR (SCHEMA, "Mode", ADA_NAME (1..LENGTH));
    when NAME_ERROR =>      -- can't find file
      OPEN_ERROR (SCHEMA, "Name", ADA_NAME (1..LENGTH));
    when USE_ERROR =>       -- can't perform requested operation
      OPEN_ERROR (SCHEMA, "Use", ADA_NAME (1..LENGTH));
    when DEVICE_ERROR =>    -- device malfunction
      OPEN_ERROR (SCHEMA, "Device", ADA_NAME (1..LENGTH));
    when END_ERROR =>       -- eof
      OPEN_ERROR (SCHEMA, "End", ADA_NAME (1..LENGTH));
    when DATA_ERROR =>      -- bad data
      OPEN_ERROR (SCHEMA, "Data", ADA_NAME (1..LENGTH));
    when LAYOUT_ERROR =>    -- page format error
      OPEN_ERROR (SCHEMA, "Layout", ADA_NAME (1..LENGTH));

  end OPEN_SCHEMA_UNIT;

------------------------------------------------------------------------
--
-- GET_STRING

  procedure GET_STRING
           (SCHEMA   : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            STR      : out STRING;
            LAST     : out NATURAL) is

    TOKEND : POSITIVE := 1;
    TLAST  : POSITIVE := 1;

  begin
    if SCHEMA.SCHEMA_STATUS = PROCESSING or else
                  SCHEMA.SCHEMA_STATUS = WITHING then
      TOKEN_END (SCHEMA, TOKEND);
      TLAST  := STR'FIRST + TOKEND - SCHEMA.STREAM.NEXT;
      STR (STR'FIRST..TLAST) := SCHEMA.STREAM.BUFFER
                               (SCHEMA.STREAM.NEXT..TOKEND);
      LAST := TLAST;
      SCHEMA.STREAM.START := SCHEMA.STREAM.NEXT;
      SCHEMA.STREAM.NEXT := TOKEND + 1;
    end if;
  end GET_STRING;

------------------------------------------------------------------------
--
-- CLOSE_SCHEMA_UNIT
--

  procedure CLOSE_SCHEMA_UNIT
```

```
          (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
begin
  if DEBUGGING then
    PRINT_TO_FILE ("*** Closing schema unit: " & STRING(SCHEMA.NAME.all) &
                   DOT_ADA_DEFAULT);
  end if;
  if WHERE_IS_SCHEMA_FROM = CALLS and STRING(SCHEMA.NAME.all) =
                  SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN) then
    null;
  else
    CLOSE (SCHEMA.STREAM.FILE);
  end if;
  SCHEMA.SCHEMA_STATUS := DONE;

exception
  when STATUS_ERROR =>   -- reading unopen file, opening open file
    CLOSE_ERROR (SCHEMA, "Status", SCHEMA.NAME.all);
  when MODE_ERROR =>     -- read output or write input
    CLOSE_ERROR (SCHEMA, "Mode", SCHEMA.NAME.all);
  when NAME_ERROR =>     -- can't find file
    CLOSE_ERROR (SCHEMA, "Name", SCHEMA.NAME.all);
  when USE_ERROR =>      -- can't perform requested operation
    CLOSE_ERROR (SCHEMA, "Use", SCHEMA.NAME.all);
  when DEVICE_ERROR =>   -- device malfunction
    CLOSE_ERROR (SCHEMA, "Device", SCHEMA.NAME.all);
  when END_ERROR =>      -- eof
    CLOSE_ERROR (SCHEMA, "End", SCHEMA.NAME.all);
  when DATA_ERROR =>     -- bad data
    CLOSE_ERROR (SCHEMA, "Data", SCHEMA.NAME.all);
  when LAYOUT_ERROR =>   -- page format error
    CLOSE_ERROR (SCHEMA, "Layout", SCHEMA.NAME.all);
end CLOSE_SCHEMA_UNIT;

----------------------------------------------------------------------------
--
-- PRINT ERROR

procedure PRINT_ERROR
          (MESSAGE : in STRING) is
begin
  FATAL_ERRORS := FATAL_ERRORS + 1;
  if CURRENT_SCHEMA_UNIT /= null and then
        CURRENT_SCHEMA_UNIT.STREAM /= null and then
        CURRENT_SCHEMA_UNIT.STREAM.LINE > 0 then
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("ERROR: Schema unit " &
                   STRING (CURRENT_SCHEMA_UNIT.NAME.all) &
                   "  error on line number " &
                   NATURAL'IMAGE(CURRENT_SCHEMA_UNIT.STREAM.LINE));
```

```
      PRINT_TO_FILE (CURRENT_SCHEMA_UNIT.STREAM.ORIG_BUF
                    (1..CURRENT_SCHEMA_UNIT.STREAM.LAST));
    end if;
    PRINT_TO_FILE (MESSAGE);
  end PRINT_ERROR;

---------------------------------------------------------------------------
--
-- PRINT TO FILE

  procedure PRINT_TO_FILE
           (MESSAGE : in STRING) is
  begin
    if OUTPUT_FILE_IS_OPEN then
--      PUT_LINE (OUTPUT_FILE_TYPE, MESSAGE);
      LEXICAL_ANALYZER.REPORT_DDL_ERROR (MESSAGE);
    else
      PRINT_MESSAGE (MESSAGE);
    end if;
  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
      PRINT_ERROR_ERROR ("Status");
    when MODE_ERROR =>     -- read output or write input
      PRINT_ERROR_ERROR ("Mode");
    when NAME_ERROR =>     -- can't find file
      PRINT_ERROR_ERROR ("Name");
    when USE_ERROR =>      -- can't perform requested operation
      PRINT_ERROR_ERROR ("Use");
    when DEVICE_ERROR =>   -- device malfunction
      PRINT_ERROR_ERROR ("Device");
    when END_ERROR =>      -- eof
      PRINT_ERROR_ERROR ("End");
    when DATA_ERROR =>     -- bad data
      PRINT_ERROR_ERROR ("Data");
    when LAYOUT_ERROR =>   -- page format error
      PRINT_ERROR_ERROR ("Layout");
  end PRINT_TO_FILE;

---------------------------------------------------------------------------
--
-- PRINT MESSAGE

  procedure PRINT_MESSAGE
           (MESSAGE : in STRING) is
  begin
    PUT_LINE (MESSAGE);

  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
```

```
      PRINT_MESSAGE_ERROR ("Status");
    when MODE_ERROR =>      -- read output or write input
      PRINT_MESSAGE_ERROR ("Mode");
    when NAME_ERROR =>      -- can't find file
      PRINT_MESSAGE_ERROR ("Name");
    when USE_ERROR =>       -- can't perform requested operation
      PRINT_MESSAGE_ERROR ("Use");
    when DEVICE_ERROR =>    -- device malfunction
      PRINT_MESSAGE_ERROR ("Device");
    when END_ERROR =>       -- eof
      PRINT_MESSAGE_ERROR ("End");
    when DATA_ERROR =>      -- bad data
      PRINT_MESSAGE_ERROR ("Data");
    when LAYOUT_ERROR =>    -- page format error
      PRINT_MESSAGE_ERROR ("Layout");
  end PRINT_MESSAGE;

---------------------------------------------------------------------------
--
-- GET_TERMINAL_INPUT

  procedure GET_TERMINAL_INPUT
          (MESSAGE : in out STRING;
           LENGTH  : in out NATURAL) is

    LEN : NATURAL := 0;

  begin
    GET_LINE (MESSAGE, LENGTH);
    UPPER_CASE (MESSAGE (1..LENGTH));
  exception
    when STATUS_ERROR =>    -- reading unopen file, opening open file
      INPUT_ERROR ("Status");
    when MODE_ERROR =>      -- read output or write input
      INPUT_ERROR ("Mode");
    when NAME_ERROR =>      -- can't find file
      INPUT_ERROR ("Name");
    when USE_ERROR =>       -- can't perform requested operation
      INPUT_ERROR ("Use");
    when DEVICE_ERROR =>    -- device malfunction
      INPUT_ERROR ("Device");
    when END_ERROR =>       -- eof
      INPUT_ERROR ("End");
    when DATA_ERROR =>      -- bad data
      INPUT_ERROR ("Data");
    when LAYOUT_ERROR =>    -- page format error
      INPUT_ERROR ("Layout");
  end GET_TERMINAL_INPUT;
```

```
-----------------------------------------------------------------------
--
-- OPEN_OUTPUT_FILE
--

  procedure OPEN_OUTPUT_FILE
           (NAME : in STRING) is
  begin
    if not OUTPUT_FILE_IS_OPEN then
      OUTPUT_FILE_NAME_LEN := NAME'LAST + OFN_EXTEN_LEN;
      OUTPUT_FILE_NAME(1..OUTPUT_FILE_NAME_LEN) := NAME & OFN_EXTEN;
      if DEBUGGING then
        PRINT_TO_FILE ("*** Opening output file: " &
                       OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN));
      end if;
      --CREATE (OUTPUT_FILE_TYPE, OUT_FILE,
      --         OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN));
      OUTPUT_FILE_IS_OPEN := TRUE;
      if DEBUGGING then
        PRINT_TO_FILE ("*** Opened output file: " &
                       OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN));
      end if;
    end if;

  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
      OPEN_OUTPUT_FILE_ERROR ("Status", NAME);
    when MODE_ERROR =>     -- read output or write input
      OPEN_OUTPUT_FILE_ERROR ("Mode", NAME);
    when NAME_ERROR =>     -- can't find file
      OPEN_OUTPUT_FILE_ERROR ("Name", NAME);
    when USE_ERROR =>      -- can't perform requested operation
      OPEN_OUTPUT_FILE_ERROR ("Use", NAME);
    when DEVICE_ERROR =>   -- device malfunction
      OPEN_OUTPUT_FILE_ERROR ("Device", NAME);
    when END_ERROR =>      -- eof
      OPEN_OUTPUT_FILE_ERROR ("End", NAME);
    when DATA_ERROR =>     -- bad data
      OPEN_OUTPUT_FILE_ERROR ("Data", NAME);
    when LAYOUT_ERROR =>   -- page format error
      OPEN_OUTPUT_FILE_ERROR ("Layout", NAME);
  end OPEN_OUTPUT_FILE;

-----------------------------------------------------------------------
--
-- CLOSE_OUTPUT_FILE
--

  procedure CLOSE_OUTPUT_FILE is
```

```
  begin
    if OUTPUT_FILE_IS_OPEN then
      if DEBUGGING then
        PRINT_TO_FILE ("*** Closing output file: " &
                       OUTPUT_FILE_NAME(1..OUTPUT_FILE_NAME_LEN));
      end if;
      OUTPUT_FILE_IS_OPEN := FALSE;
      --CLOSE (OUTPUT_FILE_TYPE);
      if DEBUGGING then
        PRINT_TO_FILE ("*** Closed output file: " &
                       OUTPUT_FILE_NAME (1..OUTPUT_FILE_NAME_LEN));
      end if;
    end if;
  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
      CLOSE_OUTPUT_FILE_ERROR ("Status");
    when MODE_ERROR =>     -- read output or write input
      CLOSE_OUTPUT_FILE_ERROR ("Mode");
    when NAME_ERROR =>     -- can't find file
      CLOSE_OUTPUT_FILE_ERROR ("Name");
    when USE_ERROR =>      -- can't perform requested operation
      CLOSE_OUTPUT_FILE_ERROR ("Use");
    when DEVICE_ERROR =>   -- device malfunction
      CLOSE_OUTPUT_FILE_ERROR ("Device");
    when END_ERROR =>      -- eof
      CLOSE_OUTPUT_FILE_ERROR ("End");
    when DATA_ERROR =>     -- bad data
      CLOSE_OUTPUT_FILE_ERROR ("Data");
    when LAYOUT_ERROR =>   -- page format error
      CLOSE_OUTPUT_FILE_ERROR ("Layout");
  end CLOSE_OUTPUT_FILE;

-------------------------------------------------------------------------
--
-- UPPER_CASE

  procedure UPPER_CASE
           (LINE : in out STRING) is
  begin
    for I in LINE'RANGE loop
      if LINE (I) in 'a'..'z' then
        LINE (I) := CHARACTER'VAL (CHARACTER'POS (LINE (I)) - 32);
      end if;
    end loop;
  end UPPER_CASE;

-------------------------------------------------------------------------
--
-- LOWER_CASE
```

```
  procedure LOWER_CASE
           (LINE : in out STRING) is
  begin
    for I in LINE'RANGE loop
      if LINE (I) in 'A'..'Z' then
        LINE (I) := CHARACTER'VAL (CHARACTER'POS (LINE (I)) + 32);
      end if;
    end loop;
  end LOWER_CASE;

----------------------------------------------------------------------
--
-- DOUBLE_PRECISION_TO_STRING

  function DOUBLE_PRECISION_TO_STRING
          (NUM : in DOUBLE_PRECISION)
           return STRING is

    package CONVERT_FLOAT is new FLOAT_IO (DOUBLE_PRECISION);
    OUT_STRING : STRING (1..20) := (others => ' ');
    OVERFLOW   : STRING (1..5) := "*****";
    II : INTEGER range 1..20 := 1;

  begin
    if NUM <= 100_000.0 and NUM >= -100_000.0 then
      CONVERT_FLOAT.PUT (OUT_STRING, NUM, 5, 0);
    else
      CONVERT_FLOAT.PUT (OUT_STRING, NUM, 10, 1);
    end if;
    for I in 1..20 loop
      II := I;
      exit when OUT_STRING (I) /= ' ';
    end loop;
    return OUT_STRING (II..20);
  exception
    when STATUS_ERROR =>   RETURN OVERFLOW;
    when MODE_ERROR =>     RETURN OVERFLOW;
    when NAME_ERROR =>     RETURN OVERFLOW;
    when USE_ERROR =>      RETURN OVERFLOW;
    when DEVICE_ERROR =>   RETURN OVERFLOW;
    when END_ERROR =>      RETURN OVERFLOW;
    when DATA_ERROR =>     RETURN OVERFLOW;
    when LAYOUT_ERROR =>   RETURN OVERFLOW;
  end DOUBLE_PRECISION_TO_STRING;

----------------------------------------------------------------------
--
-- STRING_TO_DOUBLE_PRECISION
```

```
  procedure STRING_TO_DOUBLE_PRECISION
          (NUM_STRING  : in STRING;
           OK          : in out BOOLEAN;
           NUM         : out DOUBLE_PRECISION)  is

    package CONVERT_FLOAT is new FLOAT_IO (DOUBLE_PRECISION);
    LAST_USED : POSITIVE := 1;

  begin
    OK := FALSE;
    NUM := 0.0;
    CONVERT_FLOAT.GET (NUM_STRING, NUM, LAST_USED);
    if LAST_USED /= NUM_STRING'LAST then
      NUM := 0.0;
    else
      OK := TRUE;
    end if;
  exception
    when STATUS_ERROR =>   NUM := 0.0;
    when MODE_ERROR =>     NUM := 0.0;
    when NAME_ERROR =>     NUM := 0.0;
    when USE_ERROR =>      NUM := 0.0;
    when DEVICE_ERROR =>   NUM := 0.0;
    when END_ERROR =>      NUM := 0.0;
    when DATA_ERROR =>     NUM := 0.0;
    when LAYOUT_ERROR =>   NUM := 0.0;
  end STRING_TO_DOUBLE_PRECISION;

------------------------------------------------------------------------
--
-- EXCHANGE_FOR_ORIGINAL
--
-- given the schema and the buffer, exchange the token that has been converted
-- to upper case for the original token

  procedure EXCHANGE_FOR_ORIGINAL
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            BUF     : in out STRING;
            BUF_LEN : in out NATURAL) is

  begin
    if BUF_LEN = SCHEMA.STREAM.NEXT - SCHEMA.STREAM.START then
      BUF (BUF'FIRST..BUF_LEN) := SCHEMA.STREAM.ORIG_BUF
                        (SCHEMA.STREAM.START..SCHEMA.STREAM.NEXT-1);
    end if;
  end EXCHANGE_FOR_ORIGINAL;

------------------------------------------------------------------------
--
```

```
-- GET_SINGLE_QUOTE_STRING
--
-- on entry buf_len = 1 and buf = single quote.  Keep reading till ending quote
-- however if second character is quote and third character is quote return
-- the three. Valid is true if on return buf_len = 3 and buf(1) and buf(3) = '
-- the quoted string must be all on one line or it's an error

  procedure GET_SINGLE_QUOTE_STRING
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            BUF     : in out STRING;
            BUF_LEN : in out NATURAL;
            VALID   : out BOOLEAN) is

    PTR : NATURAL := 0;
    CNT : NATURAL := 1;

  begin
    VALID := FALSE;
    if BUF (1..BUF_LEN) /= "'" then
      return;
    end if;
      PTR := SCHEMA.STREAM.NEXT;
    loop
      if PTR > SCHEMA.STREAM.LAST then
        return;
      end if;
      CNT := CNT + 1;
      exit when SCHEMA.STREAM.BUFFER (PTR) = ''' and CNT > 2;
      exit when (SCHEMA.STREAM.BUFFER (PTR) = ''' and CNT = 2 and
           (PTR < SCHEMA.STREAM.LAST and then
            SCHEMA.STREAM.BUFFER (PTR + 1) /= '''));
      PTR := PTR + 1;
    end loop;
    BUF_LEN := CNT;
    BUF (2..BUF_LEN) := SCHEMA.STREAM.ORIG_BUF (SCHEMA.STREAM.NEXT..PTR);
    if BUF_LEN = 3 and BUF(1) = ''' and BUF (BUF_LEN) = ''' then
      VALID := TRUE;
    end if;
    SCHEMA.STREAM.NEXT := PTR + 1;
  end GET_SINGLE_QUOTE_STRING;

end SCHEMA_IO;
```

3.11.88 package ddl_subroutines_1_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES, SCHEMA_IO,
     EXTRA_DEFINITIONS, KEYWORD_ROUTINES;
use  IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES, SCHEMA_IO,
     EXTRA_DEFINITIONS, KEYWORD_ROUTINES;
```

```
package SUBROUTINES_1_ROUTINES is

  procedure SPLIT_PACKAGE_NAME
           (FULL_PACKAGE         : in STRING;
            OUTTER_PACKAGE       : in out STRING;
            OUTTER_PACKAGE_LAST  : in out NATURAL;
            INNER_PACKAGE        : in out STRING;
            INNER_PACKAGE_LAST   : in out NATURAL);

  procedure FIND_END_OF_STATEMENT
           (CURRENT_STRING : in out STRING;
            CURRENT_LAST   : in out NATURAL);

  function GOT_END_OF_STATEMENT
           (CURRENT_STRING : in STRING)
            return BOOLEAN;

  procedure GET_CONSTANT
           (VALID     : in out BOOLEAN;
            CON_STANT : in STRING;
            UPDATE    : in BOOLEAN);

  procedure GET_CONSTANT_MAYBE
           (VALID     : in out BOOLEAN;
            GOT       : in out BOOLEAN;
            CON_STANT : in STRING;
            UPDATE    : in BOOLEAN);

  procedure ADJUST_USER_SCHEMA
           (NAME   : in out STRING;
            LENGTH : in out NATURAL);

  function CHARACTER_STRINGS_MATCH
          (STRING_A : in STRING;
           STRING_B : in STRING)
           return BOOLEAN;

end SUBROUTINES_1_ROUTINES;
```

3.11.89 package ddl_subroutines_1.ada

```
package body SUBROUTINES_1_ROUTINES is

-----------------------------------------------------------------
--
-- SPLIT_PACKAGE_NAME
-- given inner package which may be two packages (inner.outter)
-- split them into two packages, if only one return as outter,
-- unless it's ADA_SQL, then it's inner
```

```
procedure SPLIT_PACKAGE_NAME
          (FULL_PACKAGE          : in STRING;
           OUTTER_PACKAGE        : in out STRING;
           OUTTER_PACKAGE_LAST   : in out NATURAL;
           INNER_PACKAGE         : in out STRING;
           INNER_PACKAGE_LAST    : in out NATURAL) is

  II : NATURAL := 0;

begin
  OUTTER_PACKAGE_LAST := 0;
  INNER_PACKAGE_LAST  := 0;
  if FULL_PACKAGE = ADA_SQL_PACK then
    INNER_PACKAGE_LAST := 7;
    INNER_PACKAGE (1..INNER_PACKAGE_LAST) := FULL_PACKAGE;
  else
    for I in FULL_PACKAGE'FIRST..FULL_PACKAGE'LAST loop
      II := I;
      if FULL_PACKAGE(I) = '.' then
        II := II - 1;
        exit;
      end if;
    end loop;
    OUTTER_PACKAGE_LAST := II;
    OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST) := FULL_PACKAGE
                                        (FULL_PACKAGE'FIRST..II);
    II := II + 2;
    if II <= FULL_PACKAGE'LAST then
      INNER_PACKAGE_LAST := FULL_PACKAGE'LAST - II + 1;
      INNER_PACKAGE (1..INNER_PACKAGE_LAST) := FULL_PACKAGE
                                        (II..FULL_PACKAGE'LAST);
    end if;
  end if;
end SPLIT_PACKAGE_NAME;

------------------------------------------------------------------
--
-- FIND_END_OF_STATEMENT
--
-- advance pointers to the semicolon at the end of the current statement
-- if we're already at the end just return, if we have to read further into
-- the line read into the current string so on output it will contain
-- a semicolon

  procedure FIND_END_OF_STATEMENT
          (CURRENT_STRING : in out STRING;
           CURRENT_LAST   : in out NATURAL) is
  begin
    loop
```

```
      exit when CURRENT_STRING (1..CURRENT_LAST) = ";";
      exit when CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE;
      GET_STRING (CURRENT_SCHEMA_UNIT, CURRENT_STRING, CURRENT_LAST);
    end loop;
  end FIND_END_OF_STATEMENT;

-----------------------------------------------------------------
--
-- GOT_END_OF_STATEMENT
--
-- check to see if we're currently pointing at the ; which is
-- the end of the line

  function GOT_END_OF_STATEMENT
           (CURRENT_STRING : in STRING)
            return BOOLEAN is
  begin
    return CURRENT_STRING = ";";
  end GOT_END_OF_STATEMENT;

-----------------------------------------------------------------
--
-- GET_CONSTANT
--
-- if the string in temp string matches the asked for constant and update is
-- true then read the next token and return valid as it was on input,
-- if string doesn't match constant return valid = false

  procedure GET_CONSTANT
           (VALID     : in out BOOLEAN;
            CON_STANT : in STRING;
            UPDATE    : in BOOLEAN) is
  begin
    if TEMP_STRING(1..TEMP_STRING_LAST) = CON_STANT then
      if UPDATE then
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      end if;
    else
      VALID := FALSE;
    end if;
  end GET_CONSTANT;

-----------------------------------------------------------------
--
-- GET_CONSTANT_MAYBE
--
-- if the string in temp string matches the asked for constant and update is
-- true then read the next token and return valid as it was on input
-- and return got as true,
```

```
-- if not return valid as entered and got as false

  procedure GET_CONSTANT_MAYBE
           (VALID     : in out BOOLEAN;
            GOT       : in out BOOLEAN;
            CON_STANT : in STRING;
            UPDATE    : in BOOLEAN) is
  begin
    if TEMP_STRING(1..TEMP_STRING_LAST) = CON_STANT then
      if UPDATE then
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      end if;
      GOT := TRUE;
    else
      GOT := FALSE;
    end if;
  end GET_CONSTANT_MAYBE;

-----------------------------------------------------------------
--
-- ADJUST_USER_SCHEMA
--
-- adjust the inputed user name to upper case, lower case or leave it as it
-- if the name input by the user has an .ADA or .A, or whatever is the
-- extention for this system as defined in ddl_io_defs, extention, remove it

  procedure ADJUST_USER_SCHEMA
           (NAME   : in out STRING;
            LENGTH : in out NATURAL) is
  begin
    case HOW_TO_DO_FILES is
      when UPPER_CASE => UPPER_CASE (NAME (1..LENGTH));
      when LOWER_CASE => LOWER_CASE (NAME (1..LENGTH));
      when AS_IS            => null;
    end case;
    if LENGTH >= DOT_ADA_LEN and then
       (NAME (LENGTH - DOT_ADA_LEN + 1 .. LENGTH) = DOT_ADA_UPPER or
        NAME (LENGTH - DOT_ADA_LEN + 1 .. LENGTH) = DOT_ADA_LOWER) then
      for I in LENGTH - DOT_ADA_LEN + 1 .. LENGTH loop
        NAME (I) := ' ';
      end loop;
      LENGTH := LENGTH - DOT_ADA_LEN;
    end if;
  end ADJUST_USER_SCHEMA;

----------------------------------------------------------------------
--
-- CHARACTER_STRINGS_MATCH
--
```

```
-- if the two strings match regardless of case return true

  function CHARACTER_STRINGS_MATCH
          (STRING_A : in STRING;
           STRING_B : in STRING)
           return BOOLEAN is

    S_A : STRING (1..STRING_A'LAST) := STRING_A;
    S_B : STRING (1..STRING_B'LAST) := STRING_B;

  begin
    UPPER_CASE (S_A);
    UPPER_CASE (S_B);
    return S_A = S_B;
  end CHARACTER_STRINGS_MATCH;

end SUBROUTINES_1_ROUTINES;
```

3.11.90 package ddl_show_spec.ada

```
with DATABASE, DDL_DEFINITIONS, DDL_VARIABLES, IO_DEFINITIONS, SCHEMA_IO;
use  DATABASE, DDL_DEFINITIONS, DDL_VARIABLES, IO_DEFINITIONS, SCHEMA_IO;

package SHOW_ROUTINES is

  procedure SHOW_DATA;

  procedure SHOW_SCHEMA_UNITS;

  procedure SHOW_IDENTIFIERS;

  procedure SHOW_RECORD
            (TYP : in ACCESS_RECORD_DESCRIPTOR);

  procedure SHOW_ENUMERATION
            (TYP : in ACCESS_ENUMERATION_DESCRIPTOR);

  procedure SHOW_INTEGER
            (TYP : in ACCESS_INTEGER_DESCRIPTOR);

  procedure SHOW_FLOAT
            (TYP : in ACCESS_FLOAT_DESCRIPTOR);

  procedure SHOW_STRING
            (TYP : in ACCESS_STRING_DESCRIPTOR);

  procedure SHOW_POINTERS;

  procedure SHOW_ENUMS;
```

```
end SHOW_ROUTINES;
```

3.11.91 package ddl_show.ada

```
package body SHOW_ROUTINES is

------------------------------------------------------------------
--
-- SHOW_DATA
--
-- display the schema units,

  procedure SHOW_DATA is
  begin
    SHOW_SCHEMA_UNITS;
    SHOW_IDENTIFIERS;
    SHOW_POINTERS;
    SHOW_ENUMS;
  end SHOW_DATA;

------------------------------------------------------------------
--
-- SHOW_SCHEMA_UNITS
--
-- display the schema units processed

  procedure SHOW_SCHEMA_UNITS is

    SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR := FIRST_SCHEMA_UNIT;
    WITHED : ACCESS_WITHED_UNIT_DESCRIPTOR := null;
    USED   : ACCESS_USED_PACKAGE_DESCRIPTOR := null;
    PACK   : ACCESS_DECLARED_PACKAGE_DESCRIPTOR := null;

  begin
    PRINT_TO_FILE ("************************************" &
                   "************************************");
    while SCHEMA /= null loop
      PRINT_TO_FILE (" ");
      PRINT_TO_FILE ("SCHEMA UNIT: " & STRING (SCHEMA.NAME.all));
      if SCHEMA.AUTH_ID /= null then
        PRINT_TO_FILE ("auth id: " & STRING (SCHEMA.AUTH_ID.all));
      else
        PRINT_TO_FILE ("auth id: none");
      end if;
      PRINT_TO_FILE ("authorization package: " &
                     BOOLEAN'IMAGE (SCHEMA.IS_AUTH_PACKAGE));
      PRINT_TO_FILE ("declared types: " &
                     BOOLEAN'IMAGE (SCHEMA.HAS_DECLARED_TYPES));
      PRINT_TO_FILE ("declared tables: " &
                     BOOLEAN'IMAGE (SCHEMA.HAS_DECLARED_TABLES));
```

```
      PRINT_TO_FILE ("declared variables: " &
                     BOOLEAN'IMAGE (SCHEMA.HAS_DECLARED_VARIABLES));
      WITHED := SCHEMA.FIRST_WITHED;
      while WITHED /= null loop
        PRINT_TO_FILE ("withed schema unit: " &
                     STRING (WITHED.SCHEMA_UNIT.NAME.all));
        WITHED := WITHED.NEXT_WITHED;
      end loop;
      USED := SCHEMA.FIRST_USED;
      while USED /= null loop
        PRINT_TO_FILE ("used package name: " & STRING (USED.NAME.all));
        USED := USED.NEXT_USED;
      end loop;
      PACK := SCHEMA.FIRST_DECLARED_PACKAGE;
      while PACK /= null loop
        PRINT_TO_FILE ("declared package: " & STRING (PACK.NAME.all) &
                     " end found: " & BOOLEAN'IMAGE (PACK.FOUND_END));
        PACK := PACK.NEXT_DECLARED;
      end loop;
      if SCHEMA.STREAM /= null then
        PRINT_TO_FILE ("lines processed: " &
                     NATURAL'IMAGE (SCHEMA.STREAM.LINE));
      else
        PRINT_TO_FILE ("lines processed: 0  - unit has no stream");
      end if;
      PRINT_TO_FILE ("schema status: " &
                   STATUS_SCHEMA'IMAGE(SCHEMA.SCHEMA_STATUS));
      SCHEMA := SCHEMA.NEXT_SCHEMA_UNIT;
    end loop;
  end SHOW_SCHEMA_UNITS;

-----------------------------------------------------------------
--
-- SHOW_IDENTIFIERS
--
-- display the identifiers processed

  procedure SHOW_IDENTIFIERS is

    IDENT : ACCESS_IDENTIFIER_DESCRIPTOR := FIRST_IDENTIFIER;
    FULL  : ACCESS_FULL_NAME_DESCRIPTOR := null;
    TYP   : ACCESS_TYPE_DESCRIPTOR := null;

  begin
    PRINT_TO_FILE ("*************************************" &
                 "*************************************");
    while IDENT /= null loop
      PRINT_TO_FILE (" ");
      PRINT_TO_FILE ("IDENTIFIER: " & STRING (IDENT.NAME.all));
```

```
        FULL := IDENT.FIRST_FULL_NAME;
        while FULL /= null loop
          PRINT_TO_FILE (" ");
          if FULL.TABLE_NAME = null then
            PRINT_TO_FILE ("full package name: " &
                           STRING (FULL.FULL_PACKAGE_NAME.all) &
                           "   table name: null" & "   name: " &
                           STRING (FULL.NAME.all));
          else
            PRINT_TO_FILE ("full package name: " &
                           STRING (FULL.FULL_PACKAGE_NAME.all) &
                           "   table name: " &
                           STRING (FULL.TABLE_NAME.all) & "   name: " &
                           STRING (FULL.NAME.all));
          end if;
          PRINT_TO_FILE ("      is not null: " &
               BOOLEAN'IMAGE (FULL.IS_NOT_NULL) & " is not null unique: " &
               BOOLEAN'IMAGE (FULL.IS_NOT_NULL_UNIQUE));
          PRINT_TO_FILE ("from schema unit: " &
                         STRING (FULL.SCHEMA_UNIT.NAME.all));
          TYP := FULL.TYPE_IS;
          case TYP.WHICH_TYPE is
            when REC_ORD     => SHOW_RECORD (TYP);
            when ENUMERATION => SHOW_ENUMERATION (TYP);
            when INT_EGER    => SHOW_INTEGER (TYP);
            when FL_OAT      => SHOW_FLOAT (TYP);
            when STR_ING     => SHOW_STRING (TYP);
          end case;
          FULL := FULL.NEXT_NAME;
        end loop;
        IDENT := IDENT.NEXT_IDENT;
      end loop;
    end SHOW_IDENTIFIERS;

------------------------------------------------------------------
--
-- SHOW_RECORD
--
-- display the information on a record

    procedure SHOW_RECORD
              (TYP : in ACCESS_RECORD_DESCRIPTOR) is

      COMP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_COMPONENT;
      STYP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_SUBTYPE;
      DERV : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_DERIVED;

    begin
      PRINT_TO_FILE (TYPE_TYPE'IMAGE (TYP.WHICH_TYPE) & "     " &
```

```
          KIND_TYPE'IMAGE (TYP.TYPE_KIND));
    if TYP.BASE_TYPE /= null then
      PRINT_TO_FILE ("   our base type: " &
                     STRING (TYP.BASE_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                     &  "." &
                     STRING (TYP.BASE_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.ULT_PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our ultimate parent type: " &
                     STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.
                             FULL_PACKAGE_NAME.all)
                     &  "." &
                     STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our parent: " &
                     STRING (TYP.PARENT_TYPE.FULL_NAME.
                             FULL_PACKAGE_NAME.all) &  "." &
                     STRING (TYP.PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    while STYP /= null loop
      PRINT_TO_FILE ("   subtype: " &
                     STRING (STYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (STYP.FULL_NAME.NAME.all));
      STYP := STYP.NEXT_ONE;
    end loop;
    while COMP /= null loop
      PRINT_TO_FILE ("   component: " &
                     STRING (COMP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (COMP.FULL_NAME.NAME.all));
      COMP := COMP.NEXT_ONE;
    end loop;
    while DERV /= null loop
      PRINT_TO_FILE ("   derived: " &
                     STRING (DERV.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (DERV.FULL_NAME.NAME.all));
      DERV := DERV.NEXT_ONE;
    end loop;
  end SHOW_RECORD;

------------------------------------------------------------------
--
-- SHOW_ENUMERATION
--
-- display the information on an enumeration

  procedure SHOW_ENUMERATION
          (TYP : in ACCESS_ENUMERATION_DESCRIPTOR) is
```

```
    LIT  : ACCESS_LITERAL_DESCRIPTOR := TYP.FIRST_LITERAL;
    COMP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_COMPONENT;
    STYP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_SUBTYPE;
    DERV : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_DERIVED;

  begin
    PRINT_TO_FILE (TYPE_TYPE'IMAGE (TYP.WHICH_TYPE) & "    " &
            KIND_TYPE'IMAGE (TYP.TYPE_KIND));
    if TYP.BASE_TYPE /= null then
      PRINT_TO_FILE ("   our base type: " &
                     STRING (TYP.BASE_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                     &  "." &
                     STRING (TYP.BASE_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.ULT_PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our ultimate parent type: " &
                     STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.
                             FULL_PACKAGE_NAME.all)
                     &  "." &
                     STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our parent: " &
                     STRING (TYP.PARENT_TYPE.FULL_NAME.
                             FULL_PACKAGE_NAME.all) &  "." &
                     STRING (TYP.PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    while STYP /= null loop
      PRINT_TO_FILE ("   subtype: " &
                     STRING (STYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (STYP.FULL_NAME.NAME.all));
      STYP := STYP.NEXT_ONE;
    end loop;
    while COMP /= null loop
      PRINT_TO_FILE ("   component: " &
                     STRING (COMP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (COMP.FULL_NAME.NAME.all));
      COMP := COMP.NEXT_ONE;
    end loop;
    while DERV /= null loop
      PRINT_TO_FILE ("   derived: " &
                     STRING (DERV.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (DERV.FULL_NAME.NAME.all));
      DERV := DERV.NEXT_ONE;
    end loop;
    PRINT_TO_FILE ("   number of literals: " & INTEGER'IMAGE (TYP.LAST_POS)
                 & " max length: " & INTEGER'IMAGE (TYP.MAX_LENGTH));
    while LIT /= null loop
      PRINT_TO_FILE ("   literal position: " & NATURAL'IMAGE (LIT.POS) &
```

```
                    " name: " & STRING (LIT.NAME.all));
      exit when LIT = TYP.LAST_LITERAL;
      LIT := LIT.NEXT_LITERAL;
    end loop;
  end SHOW_ENUMERATION;

-----------------------------------------------------------------
--
-- SHOW_INTEGER
--
-- display the information on an integer

  procedure SHOW_INTEGER
            (TYP : in ACCESS_INTEGER_DESCRIPTOR) is

    COMP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_COMPONENT;
    STYP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_SUBTYPE;
    DERV : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_DERIVED;

  begin
    PRINT_TO_FILE (TYPE_TYPE'IMAGE (TYP.WHICH_TYPE) & "     " &
            KIND_TYPE'IMAGE (TYP.TYPE_KIND));
    if TYP.BASE_TYPE /= null then
      PRINT_TO_FILE ("   our base type: " &
                    STRING (TYP.BASE_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                    & "." &
                    STRING (TYP.BASE_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.ULT_PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our ultimate parent type: " &
                    STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.
                            FULL_PACKAGE_NAME.all)
                    & "." &
                    STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our parent: " &
                    STRING (TYP.PARENT_TYPE.FULL_NAME.
                            FULL_PACKAGE_NAME.all) & "." &
                    STRING (TYP.PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    while STYP /= null loop
      PRINT_TO_FILE ("   subtype: " &
                    STRING (STYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                    STRING (STYP.FULL_NAME.NAME.all));
      STYP := STYP.NEXT_ONE;
    end loop;
    while COMP /= null loop
      PRINT_TO_FILE ("   component: " &
```

```
                      STRING (COMP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                      STRING (COMP.FULL_NAME.NAME.all));
      COMP := COMP.NEXT_ONE;
    end loop;
    while DERV /= null loop
      PRINT_TO_FILE ("   derived: " &
                      STRING (DERV.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                      STRING (DERV.FULL_NAME.NAME.all));
      DERV := DERV.NEXT_ONE;
    end loop;
    PRINT_TO_FILE ("   range lo: " & INT'IMAGE (TYP.RANGE_LO_INT) &
                   "   range hi: " & INT'IMAGE (TYP.RANGE_HI_INT));
  end SHOW_INTEGER;

-----------------------------------------------------------------
--
-- SHOW_FLOAT
--
-- display the information on a float

  procedure SHOW_FLOAT
            (TYP : in ACCESS_FLOAT_DESCRIPTOR) is

    COMP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_COMPONENT;
    STYP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_SUBTYPE;
    DERV : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_DERIVED;

  begin
    PRINT_TO_FILE (TYPE_TYPE'IMAGE (TYP.WHICH_TYPE) & "    " &
            KIND_TYPE'IMAGE (TYP.TYPE_KIND));
    if TYP.BASE_TYPE /= null then
      PRINT_TO_FILE ("   our base type: " &
                      STRING (TYP.BASE_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                      &  "." &
                      STRING (TYP.BASE_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.ULT_PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our ultimate parent type: " &
                      STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.
                              FULL_PACKAGE_NAME.all)
                      &  "." &
                      STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our parent: " &
                      STRING (TYP.PARENT_TYPE.FULL_NAME.
                              FULL_PACKAGE_NAME.all) &  "." &
                      STRING (TYP.PARENT_TYPE.FULL_NAME.NAME.all));
    end if;
```

```
    while STYP /= null loop
      PRINT_TO_FILE ("   subtype: " &
                     STRING (STYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (STYP.FULL_NAME.NAME.all));
      STYP := STYP.NEXT_ONE;
    end loop;
    while COMP /= null loop
      PRINT_TO_FILE ("   component: " &
                     STRING (COMP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (COMP.FULL_NAME.NAME.all));
      COMP := COMP.NEXT_ONE;
    end loop;
    while DERV /= null loop
      PRINT_TO_FILE ("   derived: " &
                     STRING (DERV.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                     STRING (DERV.FULL_NAME.NAME.all));
      DERV := DERV.NEXT_ONE;
    end loop;
    PRINT_TO_FILE ("   digits: " & INTEGER'IMAGE (TYP.FLOAT_DIGITS) &
                   "   range lo: " & DOUBLE_PRECISION_TO_STRING
                                     (TYP.RANGE_LO_FLT) &
                   "   range hi: " & DOUBLE_PRECISION_TO_STRING
                                     (TYP.RANGE_HI_FLT));
  end SHOW_FLOAT;

-----------------------------------------------------------------
--
-- SHOW_STRING
--
-- display the information on a string

  procedure SHOW_STRING
            (TYP : in ACCESS_STRING_DESCRIPTOR) is

    COMP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_COMPONENT;
    STYP : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_SUBTYPE;
    DERV : ACCESS_TYPE_DESCRIPTOR := TYP.FIRST_DERIVED;

  begin
    PRINT_TO_FILE (TYPE_TYPE'IMAGE (TYP.WHICH_TYPE) & "     " &
            KIND_TYPE'IMAGE (TYP.TYPE_KIND));
    if TYP.BASE_TYPE /= null then
      PRINT_TO_FILE ("   our base type: " &
                     STRING (TYP.BASE_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                     & "." &
                     STRING (TYP.BASE_TYPE.FULL_NAME.NAME.all));
    end if;
    if TYP.ULT_PARENT_TYPE /= null then
      PRINT_TO_FILE ("   our ultimate parent type: " &
```

```
                    STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.
                            FULL_PACKAGE_NAME.all)
                    &  "." &
                    STRING (TYP.ULT_PARENT_TYPE.FULL_NAME.NAME.all));
  end if;
  if TYP.PARENT_TYPE /= null then
    PRINT_TO_FILE ("   our parent: " &
                    STRING (TYP.PARENT_TYPE.FULL_NAME.
                            FULL_PACKAGE_NAME.all) &  "." &
                    STRING (TYP.PARENT_TYPE.FULL_NAME.NAME.all));
  end if;
  while STYP /= null loop
    PRINT_TO_FILE ("   subtype: " &
                    STRING (STYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                    STRING (STYP.FULL_NAME.NAME.all));
    STYP := STYP.NEXT_ONE;
  end loop;
  while COMP /= null loop
    PRINT_TO_FILE ("   component: " &
                    STRING (COMP.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                    STRING (COMP.FULL_NAME.NAME.all));
    COMP := COMP.NEXT_ONE;
  end loop;
  while DERV /= null loop
    PRINT_TO_FILE ("   derived: " &
                    STRING (DERV.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
                    STRING (DERV.FULL_NAME.NAME.all));
    DERV := DERV.NEXT_ONE;
  end loop;
  PRINT_TO_FILE ("   length: " & NATURAL'IMAGE (TYP.LENGTH));
  PRINT_TO_FILE ("   constrained: " & BOOLEAN'IMAGE (TYP.CONSTRAINED));
  PRINT_TO_FILE ("   array range lo: " & INT'IMAGE (TYP.ARRAY_RANGE_LO));
  PRINT_TO_FILE ("               hi: " & INT'IMAGE (TYP.ARRAY_RANGE_HI));
  PRINT_TO_FILE ("              min: " & INT'IMAGE (TYP.ARRAY_RANGE_MIN));
  PRINT_TO_FILE ("              max: " & INT'IMAGE (TYP.ARRAY_RANGE_MAX));
  if TYP.INDEX_TYPE /= null then
    PRINT_TO_FILE ("   index type: " &
                    STRING (TYP.INDEX_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                    & "." &
                    STRING (TYP.INDEX_TYPE.FULL_NAME.NAME.all));
  end if;
  if TYP.ARRAY_TYPE /= null then
    PRINT_TO_FILE ("   array type: " &
                    STRING (TYP.ARRAY_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all)
                    & "." &
                    STRING (TYP.ARRAY_TYPE.FULL_NAME.NAME.all));
  end if;
end SHOW_STRING;
```

```
-----------------------------------------------------------------------------
--
--  SHOW_POINTERS

  procedure SHOW_POINTERS is

  YTD     : ACCESS_YET_TO_DO_DESCRIPTOR := FIRST_YET_TO_DO;
  SCHEMA  : ACCESS_SCHEMA_UNIT_DESCRIPTOR := FIRST_SCHEMA_UNIT;
  IDENT   : ACCESS_IDENTIFIER_DESCRIPTOR := FIRST_IDENTIFIER;
  FULL    : ACCESS_FULL_NAME_DESCRIPTOR := null;
  TYP     : ACCESS_TYPE_DESCRIPTOR := FIRST_TYPE;
  TAB     : ACCESS_TYPE_DESCRIPTOR := FIRST_TABLE;
  VAR     : ACCESS_TYPE_DESCRIPTOR := FIRST_VARIABLE;

  begin
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all pointers");
    if FIRST_YET_TO_DO = null then
      PRINT_TO_FILE ("Yet to do list exhausted");
    else
      while YTD /= null loop
        PRINT_TO_FILE ("Yet to do schema: " &
                       STRING (YTD.UNDONE_SCHEMA.NAME.all));
        YTD := YTD.NEXT_YET_TO_DO;
      end loop;
    end if;
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all schemas");
    while SCHEMA /= null loop
      PRINT_TO_FILE ("Schema unit: " & STRING (SCHEMA.NAME.all));
      SCHEMA := SCHEMA.NEXT_SCHEMA_UNIT;
    end loop;
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all identifiers");
    PRINT_TO_FILE ("  package  table  identifier");
    while IDENT /= null loop
      FULL := IDENT.FIRST_FULL_NAME;
      while FULL /= null loop
        if FULL.TABLE_NAME /= null then
          PRINT_TO_FILE (STRING (FULL.FULL_PACKAGE_NAME.all) & "    " &
                                 STRING (FULL.TABLE_NAME.all) & "    " &
                                 STRING (IDENT.NAME.all));
        else
          PRINT_TO_FILE (STRING (FULL.FULL_PACKAGE_NAME.all) &
                                 "   null    " &
                                 STRING (IDENT.NAME.all));
        end if;
        FULL := FULL.NEXT_NAME;
      end loop;
```

```
      IDENT := IDENT.NEXT_IDENT;
    end loop;
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all types, subtypes, deriveds, and components");
    while TYP /= null loop
      PRINT_TO_FILE (KIND_TYPE'IMAGE (TYP.TYPE_KIND) & " - " &
                     TYPE_TYPE'IMAGE (TYP.WHICH_TYPE));
      if TYP.FULL_NAME.TABLE_NAME /= null then
        PRINT_TO_FILE ("    " &
                       STRING (TYP.FULL_NAME.FULL_PACKAGE_NAME.all) & "    " &
                       STRING (TYP.FULL_NAME.TABLE_NAME.all) & "    " &
                       STRING (TYP.FULL_NAME.NAME.all));
      else
        PRINT_TO_FILE ("    " &
                       STRING (TYP.FULL_NAME.FULL_PACKAGE_NAME.all) &
                       "    null    " &
                       STRING (TYP.FULL_NAME.NAME.all));
      end if;
      TYP := TYP.NEXT_TYPE;
    end loop;
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all tables");
    while TAB /= null loop
      PRINT_TO_FILE (KIND_TYPE'IMAGE (TAB.TYPE_KIND) & " - " &
                     TYPE_TYPE'IMAGE (TAB.WHICH_TYPE));
      if TAB.FULL_NAME.TABLE_NAME /= null then
        PRINT_TO_FILE ("    " &
                       STRING (TAB.FULL_NAME.FULL_PACKAGE_NAME.all) & "    " &
                       STRING (TAB.FULL_NAME.TABLE_NAME.all) & "    " &
                       STRING (TAB.FULL_NAME.NAME.all));
      else
        PRINT_TO_FILE ("    " &
                       STRING (TAB.FULL_NAME.FULL_PACKAGE_NAME.all) &
                       "    null    " &
                       STRING (TAB.FULL_NAME.NAME.all));
      end if;
      TAB := TAB.NEXT_TYPE;
    end loop;
    PRINT_TO_FILE (" ");
    PRINT_TO_FILE ("Display of all variables");
    while VAR /= null loop
      PRINT_TO_FILE (KIND_TYPE'IMAGE (VAR.TYPE_KIND) & " - " &
                     TYPE_TYPE'IMAGE (VAR.WHICH_TYPE));
      if VAR.FULL_NAME.TABLE_NAME /= null then
        PRINT_TO_FILE ("    " &
                       STRING (VAR.FULL_NAME.FULL_PACKAGE_NAME.all) & "    " &
                       STRING (VAR.FULL_NAME.TABLE_NAME.all) & "    " &
                       STRING (VAR.FULL_NAME.NAME.all));
      else
```

```
        PRINT_TO_FILE ("    " &
                       STRING (VAR.FULL_NAME.FULL_PACKAGE_NAME.all) &
                       "   null   " &
                       STRING (VAR.FULL_NAME.NAME.all));
      end if;
      VAR := VAR.NEXT_TYPE;
    end loop;
  end SHOW_POINTERS;

-----------------------------------------------------------------
--
-- SHOW_ENUMS
--
-- display the enumeration literal chain

  procedure SHOW_ENUMS is

    LIT    : ACCESS_ENUM_LIT_DESCRIPTOR := FIRST_ENUM_LIT;
    FULL   : ACCESS_FULL_ENUM_LIT_DESCRIPTOR := null;
    TYP    : ACCESS_TYPE_DESCRIPTOR := null;

  begin
    PRINT_TO_FILE ("************************************" &
                   "************************************");
    PRINT_TO_FILE ("Display of all enumeration literals");
    while LIT /= null loop
      PRINT_TO_FILE (" ");
      PRINT_TO_FILE ("ENUM LIT: " & STRING (LIT.NAME.all));
      FULL := LIT.FIRST_FULL_ENUM_LIT;
      while FULL /= null loop
        PRINT_TO_FILE (" ");
        if FULL.TYPE_IS.FULL_NAME.TABLE_NAME = null then
          PRINT_TO_FILE ("full package name: " &
               STRING (FULL.TYPE_IS.FULL_NAME.FULL_PACKAGE_NAME.all) &
               "   table name: null" & "   lit: " &
               STRING (FULL.TYPE_IS.FULL_NAME.NAME.all));
        else
          PRINT_TO_FILE ("full package name: " &
               STRING (FULL.TYPE_IS.FULL_NAME.FULL_PACKAGE_NAME.all) &
               "   table name: " &
               STRING (FULL.TYPE_IS.FULL_NAME.TABLE_NAME.all) & "   lit: " &
               STRING (FULL.TYPE_IS.FULL_NAME.NAME.all));
        end if;
        FULL := FULL.NEXT_LIT;
      end loop;
      LIT := LIT.NEXT_ENUM_LIT;
    end loop;
  end SHOW_ENUMS;
```

```
end SHOW_ROUTINES;
```

3.11.92 package ddl_schema_io_internal.ada

```
with SCHEMA_IO;
use  SCHEMA_IO;
package body IO_INTERNAL_STUFF is

-----------------------------------------------------------------------------
--
-- TOKEN_END
--
-- point to beginning of token to read,  there are two possible cases for us
-- to read.  One is an alpha type - this must start with A .. Z and then may
-- be followed with A..Z 0..9 _ or . No further rules apply except to the .
-- which is assumed to be qualifying something.  If the . if the first
-- character it gets returned separately.  it must be followed by A..Z
-- not any thing else.  if two dots are found in a row we return up to
-- but not including the first one
-- the other type is numeric - it starts with a + or - or 0..9 then is
-- followed by 0..9 or _ and maybe an E.  After hitting an E we have to
-- have + or - or 0..9 and then only 0..9 or _ the rest of the token

  procedure TOKEN_END
           (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            T_END  : out POSITIVE) is

    C      : CHARACTER := ' ';
    PTR    : POSITIVE := 1;
    DOT    : BOOLEAN := FALSE;
    OK     : BOOLEAN := FALSE;
    EXP    : NATURAL := 0;

  begin
    NEXT_TOKEN (SCHEMA);
    PTR := SCHEMA.STREAM.NEXT;
    while PTR <= SCHEMA.STREAM.LAST loop
      C := SCHEMA.STREAM.BUFFER (PTR);
      exit when WHITESPACE (C);
      case SCHEMA.STREAM.BUFFER (SCHEMA.STREAM.NEXT) is
      when 'A'..'Z' | 'a'..'z' =>
        exit when not ALPHABETIC (C) and then not SIMPLE_NUMERIC (C)
               and then not QUALIFIER (C, SCHEMA.STREAM.BUFFER, PTR,
                                    SCHEMA.STREAM.NEXT, SCHEMA.STREAM.LAST);
      when '0'..'9' | '-' | '+' =>
          NUMERIC (OK, C, DOT, EXP, PTR, SCHEMA.STREAM.NEXT,
                   SCHEMA.STREAM.LAST, SCHEMA.STREAM.BUFFER);
        exit when not OK;
      when others =>
        exit; -- when ALPHABETIC (C) or else SIMPLE_NUMERIC (C);
```

```
      end case;
      PTR := PTR + 1;
    end loop;
    if PTR > SCHEMA.STREAM.NEXT then
      T_END := PTR - 1;
    else
      T_END := SCHEMA.STREAM.NEXT;
    end if;
  end TOKEN_END;

------------------------------------------------------------------------
--
-- WHITESPACE

  function WHITESPACE
          (C : in CHARACTER)
           return BOOLEAN is
  begin
    return C = ' ' or else C = ASCII.HT;
  end WHITESPACE;

------------------------------------------------------------------------
--
-- ALPHABETIC

  function ALPHABETIC
          (C : in CHARACTER)
           return BOOLEAN is
  begin
    return C in 'A'..'Z' or else C in 'a'..'z' or else C = '_';
  end ALPHABETIC;

------------------------------------------------------------------------
--
-- SIMPLE_NUMERIC

  function SIMPLE_NUMERIC
          (C : in CHARACTER)
           return BOOLEAN is
  begin
    return C in '0'..'9' or else C = '_';
  end SIMPLE_NUMERIC;

------------------------------------------------------------------------
--
-- QUALIFIER
--
-- C is the character in question and if it's not a dot it certainly isn't
-- a qualifier here.  Then if the next character is A..Z it's ok
```

```
  function QUALIFIER
          (C      : in CHARACTER;
           BUF    : in STRING;
           PTR    : in NATURAL;
           FIRST  : in POSITIVE;
           LAST   : in NATURAL)
           return BOOLEAN is
  begin
    return C = '.' and then PTR > FIRST and then PTR < LAST and then
           (BUF (PTR+1) in 'A'..'Z' or else BUF (PTR+1) in 'a'..'z');
  end QUALIFIER;

-----------------------------------------------------------------------------
--
-- NUMERIC

  procedure NUMERIC
          (OK     : out BOOLEAN;
           C      : in CHARACTER;
           DOT    : in out BOOLEAN;
           EXP    : in out NATURAL;
           PTR    : in NATURAL;
           FIRST  : in POSITIVE;
           LAST   : in POSITIVE;
           BUF    : in STRING) is
  begin
    OK := FALSE;
    case C is
      when '0'..'9' | '_' =>
        OK := TRUE;
      when '+' | '-' =>
        if PTR = FIRST or else (EXP > 0 and then PTR = EXP+1) then
          OK := TRUE;
        end if;
      when '.' =>
        if EXP = 0 and then
           DOT = FALSE and then
           ((PTR = LAST) or
            (PTR < LAST and then VALID_AFTER_DECIMAL (BUF (PTR + 1)))) then
          OK := TRUE;
          DOT := TRUE;
        end if;
      when 'E' =>
        if EXP = 0 then
          EXP := PTR;
          OK := TRUE;
        end if;
      when others => null;
    end case;
```

```
  end NUMERIC;

------------------------------------------------------------------------------
--
-- VALID_AFTER_DECIMAL

  function VALID_AFTER_DECIMAL
          (C : in CHARACTER)
           return BOOLEAN is
  begin
    return ((WHITESPACE (C)) or
            (SIMPLE_NUMERIC (C) and C /= '_') or
            (C = 'E') or
            (C = ')'));
  end VALID_AFTER_DECIMAL;

------------------------------------------------------------------------------
--
-- NEXT_TOKEN
--
-- we want to end up pointing at the beginning of the next token, it could
-- already be there
-- if we've reached the end of the line or a comment, read the next line
-- skip leading spaces and horizontal tabs

  procedure NEXT_TOKEN
           (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
  begin
    loop
      if SCHEMA.STREAM.NEXT > SCHEMA.STREAM.LAST then
      NEXT_LINE (SCHEMA);
      end if;
      if SCHEMA.STREAM.BUFFER (SCHEMA.STREAM.NEXT) = '-' and then
        SCHEMA.STREAM.NEXT < SCHEMA.STREAM.LAST and then
        SCHEMA.STREAM.BUFFER (SCHEMA.STREAM.NEXT+1) = '-' then
      NEXT_LINE (SCHEMA);
      end if;
      exit when SCHEMA.SCHEMA_STATUS = DONE;
      exit when SCHEMA.SCHEMA_STATUS = NOTOPEN;
      exit when SCHEMA.SCHEMA_STATUS = NOTFOUND;
      exit when not WHITESPACE (SCHEMA.STREAM.BUFFER (SCHEMA.STREAM.NEXT));
      SCHEMA.STREAM.NEXT := SCHEMA.STREAM.NEXT + 1;
    end loop;
  end NEXT_TOKEN;

------------------------------------------------------------------------------
--
-- NEXT_LINE
--
```

```
-- we read a line from the file if it's really ready to be processed
-- don't keep comment lines
-- if we get an exception - we're expecting eof sooner or later - we print
-- a message if anything other than eof and set SCHEMA.SCHEMA_STATUS to
-- DONE and close the file
-- and set schema.stream.buffer(1..2) to spaces and schema.stream.next
-- to 1 and schema.stream.last to 1.

  procedure NEXT_LINE
           (SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
  begin
    if SCHEMA.SCHEMA_STATUS = PROCESSING or
       SCHEMA.SCHEMA_STATUS = WITHING then
      loop
        if WHERE_IS_SCHEMA_FROM = CALLS and STRING (SCHEMA.NAME.all) =
                    SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN) then
          READ_ERROR (SCHEMA, "End", SCHEMA.NAME.all);
        else
          GET_LINE (SCHEMA.STREAM.FILE, SCHEMA.STREAM.ORIG_BUF,
                    SCHEMA.STREAM.LAST);
          SCHEMA.STREAM.BUFFER (1..SCHEMA.STREAM.LAST) :=
                   SCHEMA.STREAM.ORIG_BUF (1..SCHEMA.STREAM.LAST);
          SCHEMA.STREAM.LINE := SCHEMA.STREAM.LINE + 1;
        end if;
        exit when SCHEMA.STREAM.LAST >= 2 and then
                  SCHEMA.STREAM.BUFFER (1..2) /= "--";
        exit when SCHEMA.STREAM.LAST = 1;
      end loop;
      SCHEMA.STREAM.NEXT := 1;
      UPPER_CASE (SCHEMA.STREAM.BUFFER (1..SCHEMA.STREAM.LAST));
    end if;

  exception
    when STATUS_ERROR =>   -- reading unopen file, opening open file
      READ_ERROR (SCHEMA, "Status", SCHEMA.NAME.all);
    when MODE_ERROR =>     -- read output or write input
      READ_ERROR (SCHEMA, "Mode", SCHEMA.NAME.all);
    when NAME_ERROR =>     -- can't find file
      READ_ERROR (SCHEMA, "Name", SCHEMA.NAME.all);
    when USE_ERROR =>      -- can't perform requested operation
      READ_ERROR (SCHEMA, "Use", SCHEMA.NAME.all);
    when DEVICE_ERROR =>   -- device malfunction
      READ_ERROR (SCHEMA, "Device", SCHEMA.NAME.all);
    when END_ERROR =>      -- eof
      READ_ERROR (SCHEMA, "End", SCHEMA.NAME.all);
    when DATA_ERROR =>     -- bad data
      READ_ERROR (SCHEMA, "Data", SCHEMA.NAME.all);
    when LAYOUT_ERROR =>   -- page format error
      READ_ERROR (SCHEMA, "Layout", SCHEMA.NAME.all);
```

```
  end NEXT_LINE;

end IO_INTERNAL_STUFF;
```

3.11.93 package ddl_schema_io_errors.ada

```
with SCHEMA_IO;
use  SCHEMA_IO;
package body IO_ERRORS is

------------------------------------------------------------------------
--
-- OPEN_ERROR

  procedure OPEN_ERROR
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            MESSAGE : in STRING;
            NAME    : in STRING) is
  begin
    PRINT_ERROR (MESSAGE & " error - opening schema unit: " & NAME);
    SCHEMA.SCHEMA_STATUS := NOTFOUND;
    SCHEMA.STREAM.BUFFER (1..2) := "  ";
    SCHEMA.STREAM.NEXT := 1;
    SCHEMA.STREAM.LAST := 1;
  end OPEN_ERROR;

------------------------------------------------------------------------
--
-- READ_ERROR
--
-- we got an exception while reading - we're expecting eof sooner or later -
-- we print the message if anything other than eof
-- set SCHEMA.SCHEMA_STATUS to DONE
-- set schema.stream.buffer(1..2) to spaces
-- schema.stream.next to 1
-- schema.stream.last to 1.
-- close the file

  procedure READ_ERROR
           (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            MESSAGE : in STRING;
            NAME    : in LIBRARY_UNIT_NAME_STRING) is
  begin
    if MESSAGE /= "End" then
      PRINT_ERROR (MESSAGE & " error - reading from schema unit: " &
                   STRING (NAME) & DOT_ADA_DEFAULT);
    end if;
    SCHEMA.SCHEMA_STATUS := DONE;
    if DEBUGGING then
      PRINT_TO_FILE ("*** Reached eof on schema unit: " &
```

```
                        STRING(SCHEMA.NAME.all));
    end if;
    SCHEMA.STREAM.BUFFER(1..2) := "  ";
    SCHEMA.STREAM.NEXT := 1;
    SCHEMA.STREAM.LAST := 1;
    CLOSE_SCHEMA_UNIT (SCHEMA);
  end READ_ERROR;

------------------------------------------------------------------------
--
-- CLOSE_ERROR

  procedure CLOSE_ERROR
         (SCHEMA  : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR;
          MESSAGE : in STRING;
          NAME    : in LIBRARY_UNIT_NAME_STRING) is
  begin
    PRINT_ERROR (MESSAGE & " error - closing schema unit: " &
                 STRING (NAME) & DOT_ADA_DEFAULT);
  end CLOSE_ERROR;

------------------------------------------------------------------------
--
-- PRINT_ERROR_ERROR

  procedure PRINT_ERROR_ERROR
         (MESSAGE : in STRING) is
  begin
    PRINT_MESSAGE (MESSAGE & " error - writing to output file");
    CLOSE_OUTPUT_FILE;
  end PRINT_ERROR_ERROR;


------------------------------------------------------------------------
--
-- PRINT_MESSAGE_ERROR

  procedure PRINT_MESSAGE_ERROR
         (MESSAGE : in STRING) is
  begin
    PRINT_ERROR (MESSAGE & " error - writing to terminal");
  end PRINT_MESSAGE_ERROR;
------------------------------------------------------------------------
--
-- INPUT_ERROR

  procedure INPUT_ERROR
         (MESSAGE : in STRING) is
  begin
```

```
    PRINT_ERROR (MESSAGE & " error - reading from terminal");
    PRINT_MESSAGE (MESSAGE & " error - reading from terminal");
  end INPUT_ERROR;

------------------------------------------------------------------------
--
-- OPEN_OUTPUT_FILE_ERROR

  procedure OPEN_OUTPUT_FILE_ERROR
           (MESSAGE : in STRING;
            NAME    : in STRING) is
  begin
    PRINT_ERROR (MESSAGE & " error - opening output file: " & NAME);
    PRINT_MESSAGE (MESSAGE & " error - opening output file: " & NAME);
  end OPEN_OUTPUT_FILE_ERROR;

------------------------------------------------------------------------
--
-- CLOSE_OUTPUT_FILE_ERROR

  procedure CLOSE_OUTPUT_FILE_ERROR
           (MESSAGE : in STRING) is
  begin
    PRINT_ERROR (MESSAGE & " error - closing the output file");
  end CLOSE_OUTPUT_FILE_ERROR;

end IO_ERRORS;
```

3.11.94 package ddl_end_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO;

package END_ROUTINES is

  procedure PROCESS_END;

  procedure END_LAST_PACKAGE;

  procedure END_NAMED_PACKAGE
           (NAME      : in STRING;
            NAME_LAST : in NATURAL);

end END_ROUTINES;
```

3.11.95 package ddl_end.ada

```
package body END_ROUTINES is

----------------------------------------------------------------
```

```
--
-- PROCESS_END
--
-- the only end we'll get here is the end of a package, it may be followed
-- by the package name or it may be followed by just a semicolon.  If a
-- package name then it better be the last defined not yet ended since
-- if there is more than one it would have to be nested.  If it's not the
-- last one but is a match tell em out of order end but go ahead and flag
-- it as done anyway.  If it's a semi colon then it matches up to the
-- lastest one not ended.  After it's processed call set up our package name
-- to alter current package name.

  procedure PROCESS_END is

    PACKAGE_NAME      : STRING (1..250) := (others => ' ');
    PACKAGE_NAME_LAST : NATURAL := 0;
    PACK_DES          : ACCESS_DECLARED_PACKAGE_DESCRIPTOR := null;
    LAST_PACKAGE      : BOOLEAN := FALSE;

  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** END");
    end if;
    GET_STRING (CURRENT_SCHEMA_UNIT, PACKAGE_NAME, PACKAGE_NAME_LAST);
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE then
      PRINT_ERROR ("Incomplete end package declaration - no package name " &
                   "or terminating ;");
    elsif PACKAGE_NAME(1..PACKAGE_NAME_LAST) = ";" then
      LAST_PACKAGE := TRUE;
      END_LAST_PACKAGE;
    else
      END_NAMED_PACKAGE (PACKAGE_NAME, PACKAGE_NAME_LAST);
      GET_STRING (CURRENT_SCHEMA_UNIT, PACKAGE_NAME, PACKAGE_NAME_LAST);
      if PACKAGE_NAME (1..PACKAGE_NAME_LAST) /= ";" then
        PRINT_ERROR ("Incomplete end package declaration - no terminating ;");
      end if;
    end if;
  end PROCESS_END;

-----------------------------------------------------------------
--
-- END_LAST_PACKAGE
--
-- we have the end for the last unended package, the only error is if there
-- is no package to end

  procedure END_LAST_PACKAGE is

    PACK_DES          : ACCESS_DECLARED_PACKAGE_DESCRIPTOR :=
```

```
                         CURRENT_SCHEMA_UNIT.LAST_DECLARED_PACKAGE;

  begin
    while PACK_DES /= null loop
      if not PACK_DES.FOUND_END then
        PACK_DES.FOUND_END := TRUE;
        if DEBUGGING then
          PRINT_TO_FILE ("     - ending last package: " &
                         STRING (PACK_DES.NAME.all));
        end if;
        return;
      end if;
      PACK_DES := PACK_DES.PREVIOUS_DECLARED;
    end loop;
    if DEBUGGING then
      PRINT_TO_FILE ("     - attempting to end last package");
    end if;
    PRINT_ERROR ("No corresponding package declaration");
  end END_LAST_PACKAGE;

-----------------------------------------------------------------
--
-- END_NAMED_PACKAGE
--
-- we have the end for a named package, the only error is if there
-- is no package to end, or if the end is out of order since packages should
-- be nested

  procedure END_NAMED_PACKAGE
           (NAME      : in STRING;
            NAME_LAST : in NATURAL) is

    BAD_ORDER          : BOOLEAN := FALSE;
    PACK_DES           : ACCESS_DECLARED_PACKAGE_DESCRIPTOR :=

                         CURRENT_SCHEMA_UNIT.LAST_DECLARED_PACKAGE;

  begin
    while PACK_DES /= null loop
      if not PACK_DES.FOUND_END then
        if STRING (PACK_DES.NAME.all) = NAME (1..NAME_LAST) then
          PACK_DES.FOUND_END := TRUE;
          if DEBUGGING then
            PRINT_TO_FILE ("     - ended: " &
                           STRING (NAME (1..NAME_LAST)));
          end if;
          if BAD_ORDER then
            PRINT_ERROR ("Multiple packages must be nested");
          end if;
```

```
          return;
        else
          BAD_ORDER := TRUE;
        end if;
      end if;
      PACK_DES := PACK_DES.PREVIOUS_DECLARED;
    end loop;
    if DEBUGGING then
      PRINT_TO_FILE ("    - attempting to end: " &
                     STRING (NAME (1..NAME_LAST)));
    end if;
    PRINT_ERROR ("No corresponding package declaration");
  end END_NAMED_PACKAGE;

end END_ROUTINES;
```

3.11.96 package ddl_search_des_spec.ada

```
with DDL_DEFINITIONS, DDL_VARIABLES, EXTRA_DEFINITIONS, SUBROUTINES_1_ROUTINES;
use  DDL_DEFINITIONS, DDL_VARIABLES, EXTRA_DEFINITIONS, SUBROUTINES_1_ROUTINES;

package SEARCH_DESCRIPTOR_ROUTINES is

  function FIND_NEXT_YET_TO_DO_DESCRIPTOR
           return ACCESS_SCHEMA_UNIT_DESCRIPTOR;

  function FIND_SCHEMA_UNIT_DESCRIPTOR
          (NAME : in STRING)
           return  ACCESS_SCHEMA_UNIT_DESCRIPTOR;

  function DUPLICATE_WITH
          (CURRENT_SCHEMA : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           WITH_SCHEMA    : in ACCESS_SCHEMA_UNIT_DESCRIPTOR)
           return  BOOLEAN;

  function SEARCH_WITHS_TO_FIND_A_USE
          (SCHEMA    : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           NAME      : in STRING)
           return  BOOLEAN;

  function DUPLICATE_USE
          (CURRENT_SCHEMA : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           NAME           : in STRING)
           return  BOOLEAN;

  procedure GET_PACKAGE_COUNT
          (SCHEMA        : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           PACKAGE_COUNT : in out NATURAL;
           PACKAGE_OPEN  : in out NATURAL);
```

```
  function SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE
          (AUTH            : in STRING)
           return BOOLEAN;

  procedure SET_UP_OUR_PACKAGE_NAME;

end SEARCH_DESCRIPTOR_ROUTINES;
```

3.11.97 package ddl_search_des.ada

```
package body SEARCH_DESCRIPTOR_ROUTINES is

----------------------------------------------------------------------------
--
-- FIND_NEXT-YET_TO_DO_DESCRIPTOR
--
-- return a scuema unit descriptor of the next one to do
-- if LAST_YET_TO_DO is null we return null and that means every thing's
--         been done
-- otherwise LAST_YET_TO_DO becomes the one we're going to do and
--         LAST_YET_TO_DO is reset with PREVIOUS_YET_TO_DO
-- and PREVIOUS_YET_TO_DO's NEXT pointer is nullified

  function FIND_NEXT_YET_TO_DO_DESCRIPTOR
           return ACCESS_SCHEMA_UNIT_DESCRIPTOR is

    NEXT_YET_TO_DO_DESCRIPTOR : ACCESS_YET_TO_DO_DESCRIPTOR
                              := LAST_YET_TO_DO;
    RETURN_SCHEMA_UNIT : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;

  begin
    if NEXT_YET_TO_DO_DESCRIPTOR /= null then
      RETURN_SCHEMA_UNIT := NEXT_YET_TO_DO_DESCRIPTOR.UNDONE_SCHEMA;
      LAST_YET_TO_DO := LAST_YET_TO_DO.PREVIOUS_YET_TO_DO;
      if LAST_YET_TO_DO = null then
        FIRST_YET_TO_DO := null;
      else
        LAST_YET_TO_DO.NEXT_YET_TO_DO := null;
      end if;
    end if;
    SET_UP_OUR_PACKAGE_NAME;
    return RETURN_SCHEMA_UNIT;
  end FIND_NEXT_YET_TO_DO_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- FIND_SCHEMA_UNIT_DESCRIPTOR
--
-- return pointer to schema unit with given library unit name, if none then
--      return null
```

```
-- it will only been found if it has been processed or partially processed

  function FIND_SCHEMA_UNIT_DESCRIPTOR
          (NAME : in STRING)
           return  ACCESS_SCHEMA_UNIT_DESCRIPTOR is

    DESIRED_SCHEMA_UNIT_DESCRIPTOR : ACCESS_SCHEMA_UNIT_DESCRIPTOR
                                   := FIRST_SCHEMA_UNIT;
  begin
    while DESIRED_SCHEMA_UNIT_DESCRIPTOR /= null loop
      exit when CHARACTER_STRINGS_MATCH
           (STRING (DESIRED_SCHEMA_UNIT_DESCRIPTOR.NAME.all), NAME);
      DESIRED_SCHEMA_UNIT_DESCRIPTOR :=
                     DESIRED_SCHEMA_UNIT_DESCRIPTOR.NEXT_SCHEMA_UNIT;
    end loop;
    return DESIRED_SCHEMA_UNIT_DESCRIPTOR;
  end FIND_SCHEMA_UNIT_DESCRIPTOR;

----------------------------------------------------------------------------
--
-- DUPLICATE_WITH
--

-- given the current schema we're processing and the schema of the library
-- unit we're thinking about withing, tell us if we've withed this one from
-- this schema before

  function DUPLICATE_WITH
          (CURRENT_SCHEMA : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           WITH_SCHEMA    : in ACCESS_SCHEMA_UNIT_DESCRIPTOR)
           return  BOOLEAN is

    TEST_WITH : ACCESS_WITHED_UNIT_DESCRIPTOR := CURRENT_SCHEMA.FIRST_WITHED;

  begin
    if WITH_SCHEMA /= null then
      while TEST_WITH /= null loop
        if TEST_WITH.SCHEMA_UNIT = WITH_SCHEMA then
          return TRUE;
        end if;
        TEST_WITH := TEST_WITH.NEXT_WITHED;
      end loop;
    end if;
    return FALSE;
  end DUPLICATE_WITH;

----------------------------------------------------------------------------
--
-- SEARCH_WITHS_TO_FIND_A_USE
```

```
--
-- given a schema_unit_descriptor and a used package name return true if that
-- package name is that of a withed schema, false if it's not
-- this is for the case of use clause in the context where it's name must
-- match exactly that of a withed unit

  function SEARCH_WITHS_TO_FIND_A_USE
          (SCHEMA     : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           NAME       : in STRING)
           return  BOOLEAN is

   TEST_WITH : ACCESS_WITHED_UNIT_DESCRIPTOR := SCHEMA.FIRST_WITHED;

 begin
    while TEST_WITH /= null loop
      if CHARACTER_STRINGS_MATCH (STRING (TEST_WITH.SCHEMA_UNIT.NAME.all),
                                  NAME) then
        return TRUE;
      end if;
      TEST_WITH := TEST_WITH.NEXT_WITHED;
    end loop;
    return FALSE;
  end SEARCH_WITHS_TO_FIND_A_USE;

----------------------------------------------------------------------------
--
-- DUPLICATE_USE
--
-- given the current schema we're processing and the full name of a used
-- package tell us if we've used this one from this schema before


  function DUPLICATE_USE
          (CURRENT_SCHEMA : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           NAME           : in STRING)
           return  BOOLEAN is

    TEST_USE : ACCESS_USED_PACKAGE_DESCRIPTOR := CURRENT_SCHEMA.FIRST_USED;

  begin
    while TEST_USE /= null loop
      if TEST_USE.NAME.all = PACKAGE_NAME_STRING (NAME) then
        return TRUE;
      end if;
      TEST_USE := TEST_USE.NEXT_USED;
    end loop;
    return FALSE;
  end DUPLICATE_USE;
```

```
-----------------------------------------------------------------
--
-- GET_PACKAGE_COUNT
--
-- count the number of packages already declared by this schema unit
-- and the number not ended yet

  procedure GET_PACKAGE_COUNT
          (SCHEMA        : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           PACKAGE_COUNT : in out NATURAL;
           PACKAGE_OPEN  : in out NATURAL) is

      PACKS : ACCESS_DECLARED_PACKAGE_DESCRIPTOR :=
              SCHEMA.FIRST_DECLARED_PACKAGE;
  begin
    PACKAGE_COUNT := 0;
    PACKAGE_OPEN := 0;
    while PACKS /= null loop
      PACKAGE_COUNT := PACKAGE_COUNT + 1;
      if not PACKS.FOUND_END then
        PACKAGE_OPEN := PACKAGE_OPEN + 1;
      end if;
      PACKS := PACKS.NEXT_DECLARED;
    end loop;
  end GET_PACKAGE_COUNT;

-----------------------------------------------------------------
--
-- SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE
--
-- see if this authorization identifier has been declared in an
-- authorization package withed by the current schema

  function SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE
          (AUTH             : in STRING)
           return BOOLEAN is

    A_WITH : ACCESS_WITHED_UNIT_DESCRIPTOR := CURRENT_SCHEMA_UNIT.FIRST_WITHED;

  begin
    while A_WITH /= null loop
      if A_WITH.SCHEMA_UNIT.IS_AUTH_PACKAGE and then
         STRING (A_WITH.SCHEMA_UNIT.AUTH_ID.all) = AUTH then
        return TRUE;
      end if;
      A_WITH := A_WITH.NEXT_WITHED;
    end loop;
    return FALSE;
  end SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE;
```

```
-----------------------------------------------------------------------
--
-- SET_UP_OUR_PACKAGE_NAME
--
-- set up in our_package_name the package name we're in right now

  procedure SET_UP_OUR_PACKAGE_NAME is

  PACK    : ACCESS_DECLARED_PACKAGE_DESCRIPTOR := null;
  NEW_END : NATURAL := 0;

  begin
    OUR_PACKAGE_NAME_LAST := 0;
    if CURRENT_SCHEMA_UNIT /= null then
      PACK := CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE;
      while PACK /= null loop
        if not PACK.FOUND_END then
          if OUR_PACKAGE_NAME_LAST /= 0 then
            OUR_PACKAGE_NAME_LAST := OUR_PACKAGE_NAME_LAST + 1;
            OUR_PACKAGE_NAME (OUR_PACKAGE_NAME_LAST) := '.';
          end if;
          NEW_END := OUR_PACKAGE_NAME_LAST + PACK.NAME'LAST;
          OUR_PACKAGE_NAME (OUR_PACKAGE_NAME_LAST + 1 .. NEW_END) :=
                                  STRING (PACK.NAME.all);
          OUR_PACKAGE_NAME_LAST := NEW_END;
        end if;
        PACK := PACK.NEXT_DECLARED;
      end loop;
    end if;
  end SET_UP_OUR_PACKAGE_NAME;

end SEARCH_DESCRIPTOR_ROUTINES;
```

3.11.98 package ddl_error_spec.ada

```
with SCHEMA_IO, EXTRA_DEFINITIONS, SUBROUTINES_1_ROUTINES;
use  SCHEMA_IO, EXTRA_DEFINITIONS, SUBROUTINES_1_ROUTINES;

package ERROR_ROUTINES is

  procedure PROCESS_ERROR;

end ERROR_ROUTINES;
```

3.11.99 package ddl_error.ada

```
package body ERROR_ROUTINES is

-----------------------------------------------------------------
--
```

```
-- PROCESS_ERROR

  procedure PROCESS_ERROR is
  begin
    PRINT_ERROR ("Got an unknown declaration");
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
  end PROCESS_ERROR;
end ERROR_ROUTINES;
```

3.11.100 package ddl_use_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES;

package USE_ROUTINES is

  procedure PROCESS_USE;

  procedure PROCESS_USE_CONTEXT;

  procedure PROCESS_USE_NON_CONTEXT;

  procedure VALID_USE
           (SCHEMA              : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            OUTTER_PACKAGE      : in out STRING;
            OUTTER_PACKAGE_LAST : in out NATURAL;
            INNER_PACKAGE       : in STRING;
            INNER_PACKAGE_LAST  : in NATURAL;
            IS_IT_VALID         : out BOOLEAN);


  PACK                : STRING (1..250) := (others => ' ');
  PACK_LAST           : NATURAL := 0;
  OUTTER_PACKAGE      : STRING (1..250) := (others => ' ');
  OUTTER_PACKAGE_LAST : NATURAL := 0;

end USE_ROUTINES;
```

3.11.101 package ddl_use.ada

```
package body USE_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_USE
--
```

```
-- when we enter this routine the temp string will be use
-- if no withs have been done it's an error to do a use, print error and
--    skip to end of use clause
-- if no packages have been declared we're processing a context clause use
-- if a package has been declared we're processing a non context clause use

  procedure PROCESS_USE is

    CONTEXT : BOOLEAN := FALSE;

  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** USE - processing from schema: " &
                     STRING (CURRENT_SCHEMA_UNIT.NAME.all));
    end if;
    if CURRENT_SCHEMA_UNIT.FIRST_WITHED = null then
      PRINT_ERROR ("A with clause must appear before a use clause" &
                   " - use clause ignored");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    if CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE = null then
      CONTEXT := TRUE;
      if DEBUGGING then
        PRINT_TO_FILE ("    - context clause use");
      end if;
    else
      CONTEXT := FALSE;
      if DEBUGGING then
        PRINT_TO_FILE ("    - non context clause use");
      end if;
    end if;

-- we loop and read the next token, either a comma, a semicolon or package
--        to use
-- if comma - ignore it
-- if semi colon - the use statement is done and we return
-- otherwise we have a package_name to process
-- if this schema is an authorization package the only "use" permitted
--        is for schema_definition.  Anything else print an error.
-- call the appropriate routine to check it's validity and set up the
--    visibility pointers describing it, this depends on if it's a context
--    use or a non context use

    loop
      GET_STRING (CURRENT_SCHEMA_UNIT, PACK, PACK_LAST);
      exit when CURRENT_SCHEMA_UNIT.SCHEMA_STATUS >= DONE;
      if DEBUGGING then
        PRINT_TO_FILE ("    - string: " & PACK (1..PACK_LAST));
```

```
      end if;
      exit when PACK (1..PACK_LAST) = ";";
      if PACK (1..PACK_LAST) /= "," then
        if CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE and then
               PACK (1..PACK_LAST) /= SCHEMA_DEF_NAME then
          PRINT_ERROR ("The only library unit that may be used " &
                       "by an authorization package");
          PRINT_TO_FILE ("    is " & SCHEMA_DEF_NAME);
        else
          if CONTEXT then
            PROCESS_USE_CONTEXT;
          else
            PROCESS_USE_NON_CONTEXT;
          end if;
        end if;
      end if;
    end loop;
  end PROCESS_USE;

-----------------------------------------------------------------
--
-- PROCESS_USE_CONTEXT
--
-- when we enter this routine we have a package name from a context
-- clause use.  The package name must be one that was mentioned in the
-- with clause or else we print an error.  If it hasn't been used by this
-- schema before add it to the chain

  procedure PROCESS_USE_CONTEXT is

    USED_PACKAGE : ACCESS_USED_PACKAGE_DESCRIPTOR := null;

  begin
    if not SEARCH_WITHS_TO_FIND_A_USE (CURRENT_SCHEMA_UNIT,
              PACK (1..PACK_LAST)) then
      PRINT_ERROR ("Invalid use statement: package - " & PACK(1..PACK_LAST));
      PRINT_TO_FILE ("        must previously have been declared in " &
                     "a with clause");
    elsif DUPLICATE_USE (CURRENT_SCHEMA_UNIT, PACK (1..PACK_LAST)) then
      if DEBUGGING then
        PRINT_TO_FILE ("    - duplicate use");
      end if;
    else
      USED_PACKAGE := GET_NEW_USED_PACKAGE_DESCRIPTOR;
      USED_PACKAGE.NAME := GET_NEW_PACKAGE_NAME (PACK (1..PACK_LAST));
      ADD_USED_PACKAGE_DESCRIPTOR (USED_PACKAGE, CURRENT_SCHEMA_UNIT);
      if DEBUGGING then
        PRINT_TO_FILE ("    - adding use - " & PACK (1..PACK_LAST));
      end if;
```

```
    end if;
  end PROCESS_USE_CONTEXT;

------------------------------------------------------------------
--
-- PROCESS_USE_NON_CONTEXT
--
-- when we enter this routine we have a package name from a non context
-- clause use.  The package name may be qualified with a preceding package
-- name.  But two levels is the max.  The first may be anything, the second
-- if there must be ADA_SQL.  Split the use package name into outter name
-- and inner name.  This package must then be found in a with descriptor for
-- the current schema.  If it's valid and it hasn't been used by this
-- schema before add it to the chain.  If it's invalid tell the user we can't
-- find it in a withed schema or it ambiguous.

  procedure PROCESS_USE_NON_CONTEXT is

    IS_IT_VALID          : BOOLEAN := FALSE;
    INNER_PACKAGE        : STRING (1..250) := (others => ' ');
    INNER_PACKAGE_LAST   : NATURAL := 0;
    FULL_PACKAGE         : STRING (1..250) := (others => ' ');
    FULL_PACKAGE_LAST    : NATURAL := 0;
    USED_PACKAGE         : ACCESS_USED_PACKAGE_DESCRIPTOR := null;

  begin
    SPLIT_PACKAGE_NAME (PACK (1..PACK_LAST),
                        OUTTER_PACKAGE, OUTTER_PACKAGE_LAST,
                        INNER_PACKAGE, INNER_PACKAGE_LAST);
    if OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST) = ADA_SQL_PACK or else
           (INNER_PACKAGE_LAST > 0 and then
            INNER_PACKAGE (1..INNER_PACKAGE_LAST) /= ADA_SQL_PACK) then
      PRINT_ERROR ("In the case of nested packages the inner package " &
                   "must have the name ADA_SQL,");
      PRINT_TO_FILE ("the outter package must not have the name ADA_SQL");
      return;
    end if;
    VALID_USE (CURRENT_SCHEMA_UNIT, OUTTER_PACKAGE, OUTTER_PACKAGE_LAST,
           INNER_PACKAGE, INNER_PACKAGE_LAST, IS_IT_VALID);
    if not IS_IT_VALID then
      PRINT_ERROR ("Invalid use statement - cannot use package: " &
                   PACK (1..PACK_LAST));
      return;
    end if;
    FULL_PACKAGE_LAST := OUTTER_PACKAGE_LAST;
    FULL_PACKAGE (1..FULL_PACKAGE_LAST) :=
                         OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST);
    if OUTTER_PACKAGE_LAST > 0 and INNER_PACKAGE_LAST > 0 then
      FULL_PACKAGE_LAST := FULL_PACKAGE_LAST + 1;
```

```
      FULL_PACKAGE (FULL_PACKAGE_LAST) := '.';
    end if;
    FULL_PACKAGE (FULL_PACKAGE_LAST + 1..
                    FULL_PACKAGE_LAST + INNER_PACKAGE_LAST) :=
            INNER_PACKAGE (1..INNER_PACKAGE_LAST);
    FULL_PACKAGE_LAST := FULL_PACKAGE_LAST + INNER_PACKAGE_LAST;
    if not DUPLICATE_USE (CURRENT_SCHEMA_UNIT, FULL_PACKAGE
           (1..FULL_PACKAGE_LAST)) then
      USED_PACKAGE := GET_NEW_USED_PACKAGE_DESCRIPTOR;
      USED_PACKAGE.NAME := GET_NEW_PACKAGE_NAME
                             (FULL_PACKAGE (1..FULL_PACKAGE_LAST));
      ADD_USED_PACKAGE_DESCRIPTOR (USED_PACKAGE, CURRENT_SCHEMA_UNIT);
      if DEBUGGING then
        PRINT_TO_FILE ("    - adding use: " &
                       FULL_PACKAGE (1..FULL_PACKAGE_LAST));
      end if;
    else
      if DEBUGGING then
        PRINT_TO_FILE ("    - duplicate use: " &
                       FULL_PACKAGE (1..FULL_PACKAGE_LAST));
      end if;
    end if;
  end PROCESS_USE_NON_CONTEXT;

-----------------------------------------------------------------
--
-- VALID_USE
--
-- given an outter package name and/or an inner package name and a schema unit
-- descriptor find out if these package names are valid for a use clause.
--
-- We read the withed schemas for the current schema
-- if we have an outter package and it does match but we don't have an inner,
-- or we do have an inner and it matches too, count it as a match
-- if we don't have an outter but the inner matches and this withed
--    outter package was used in our schema, count it as a match, and save
--    the outter name for later

  procedure VALID_USE
           (SCHEMA              : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
            OUTTER_PACKAGE      : in out STRING;
            OUTTER_PACKAGE_LAST : in out NATURAL;
            INNER_PACKAGE       : in STRING;
            INNER_PACKAGE_LAST  : in NATURAL;
            IS_IT_VALID         : out BOOLEAN) is

    type IS_IT_IN_OR_OUT is (INNER_ONLY, OUTTER_ONLY, BOTH, NONE);
    INNER_OR_OUTTER  : IS_IT_IN_OR_OUT := NONE;
    GOT_OUTTER       : BOOLEAN := FALSE;
```

```
    GOT_INNER           : BOOLEAN := FALSE;
    GOT_OUTTER_MATCH    : BOOLEAN := FALSE;
    GOT_INNER_MATCH     : BOOLEAN := FALSE;
    A_WITHED            : ACCESS_WITHED_UNIT_DESCRIPTOR  := SCHEMA.FIRST_WITHED;
    START_USED          : ACCESS_USED_PACKAGE_DESCRIPTOR := SCHEMA.FIRST_USED;
    A_USED              : ACCESS_USED_PACKAGE_DESCRIPTOR := SCHEMA.FIRST_USED;
    MATCH_COUNT         : NATURAL := 0;
    HOLD_OUTTER         : STRING (1..250) := (others => ' ');
    HOLD_OUTTER_LAST    : NATURAL := 0;

  begin

-- first determine if we have an inner package or outter package or both or
-- neither - if neither it's an error

    IS_IT_VALID := FALSE;
    if OUTTER_PACKAGE_LAST > 0 and INNER_PACKAGE_LAST > 0 then
      INNER_OR_OUTTER := BOTH;
      GOT_INNER := TRUE;
      GOT_OUTTER := TRUE;
    elsif OUTTER_PACKAGE_LAST > 0 then
      INNER_OR_OUTTER := OUTTER_ONLY;
      GOT_OUTTER := TRUE;
    elsif INNER_PACKAGE_LAST > 0 then
      INNER_OR_OUTTER := INNER_ONLY;
      GOT_INNER := TRUE;
    end if;
    if DEBUGGING then
      PRINT_TO_FILE ("    - got inner or outter package: " &
                     IS_IT_IN_OR_OUT'IMAGE (INNER_OR_OUTTER));
      PRINT_TO_FILE ("    - outter.inner: " &
                     OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST) & "." &
                     INNER_PACKAGE (1..INNER_PACKAGE_LAST) & ":");
    end if;
    if INNER_OR_OUTTER = NONE then
      return;
    end if;

-- loop thru all the packages withed by this schema unit and check for matches
-- if the first declared package of a schema unit matches the outter package
--    we match on outter
-- if the next declared package of the schema unit matches the inner package
--    we match on inner

    while A_WITHED /= null loop
      GOT_OUTTER_MATCH := FALSE;
      GOT_INNER_MATCH := FALSE;
      if A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /= null and then
         STRING (A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.NAME.all) =
```

```
          OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST) then
         GOT_OUTTER_MATCH := TRUE;
       end if;
       if A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /= null and then
          A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.NEXT_DECLARED /= null
          and then STRING
          (A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.NEXT_DECLARED.NAME.all) =
          INNER_PACKAGE (1..INNER_PACKAGE_LAST) then
         GOT_INNER_MATCH := TRUE;
       end if;
       if DEBUGGING then
         PRINT_TO_FILE ("     - wihted: " &
                        STRING (A_WITHED.SCHEMA_UNIT.NAME.all) &
                        " matches outter: " &
                        BOOLEAN'IMAGE (GOT_OUTTER_MATCH) & " matches inner: "
                        & BOOLEAN'IMAGE (GOT_INNER_MATCH));
       end if;

-- if we have an outter and an inner and both match, that counts as a match
-- if we have an outter and it matches and we have no inner, that counts as
--    a match

       if GOT_OUTTER and GOT_OUTTER_MATCH then
         if GOT_INNER and GOT_INNER_MATCH then
           MATCH_COUNT := MATCH_COUNT + 1;
         elsif not GOT_INNER then
           MATCH_COUNT := MATCH_COUNT + 1;
         end if;
       end if;

-- if we don't have an outter but the inner matches we check to see if the
--    outter was previously used by this schema.  If so that counts as a
--    match and we hang on to the outter name for later use

       if not GOT_OUTTER then
         if GOT_INNER_MATCH then
           A_USED := START_USED;
           while A_USED /= null loop
             if A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /= null and then
                A_USED.NAME.all =
                    A_WITHED.SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.NAME.all then
               MATCH_COUNT := MATCH_COUNT + 1;
               HOLD_OUTTER_LAST := A_USED.NAME'LAST;
               HOLD_OUTTER (1..HOLD_OUTTER_LAST) := STRING (A_USED.NAME.all);
               if DEBUGGING then
                 PRINT_TO_FILE ("     - used outter: " &
                                HOLD_OUTTER (1..HOLD_OUTTER_LAST));
               end if;
             end if;
```

```
          A_USED := A_USED.NEXT_USED;
        end loop;
      end if;
    end if;
    A_WITHED := A_WITHED.NEXT_WITHED;
  end loop;

-- if we matched one and only one package from a withed unit it's valid
-- if we're missing the outter package we stuff it into the holder

  if MATCH_COUNT = 1 then
    IS_IT_VALID := TRUE;
    if not GOT_OUTTER then
      OUTTER_PACKAGE_LAST := HOLD_OUTTER_LAST;
      OUTTER_PACKAGE (1..OUTTER_PACKAGE_LAST) :=
                               HOLD_OUTTER (1..HOLD_OUTTER_LAST);
    end if;
  elsif DEBUGGING then
    PRINT_TO_FILE ("    - ambigous # of matches " &
                   NATURAL'IMAGE (MATCH_COUNT));
  end if;
end VALID_USE;

end USE_ROUTINES;
```

3.11.102 package ddl_subroutines_2_spec.ada

```
with DATABASE, IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES,
     EXTRA_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES, KEYWORD_ROUTINES;
use  DATABASE, IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES,
     EXTRA_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES, KEYWORD_ROUTINES;

package SUBROUTINES_2_ROUTINES is

  procedure SPLIT_IDENT_2_PACKS
           (NAME             : in STRING;
            NAME_LAST        : in NATURAL;
            IDENT            : in out STRING;
            IDENT_LAST       : in out NATURAL;
            PACK1            : in out STRING;
            PACK1_LAST       : in out NATURAL;
            PACK2            : in out STRING;
            PACK2_LAST       : in out NATURAL;
            OK               : in out BOOLEAN;
            ERR_MSG          : in BOOLEAN);

  function FIND_IDENTIFIER_DESCRIPTOR
          (IDENTIFIER : in STRING)
```

```
            return ACCESS_IDENTIFIER_DESCRIPTOR;

function FIND_FULL_NAME_DESCRIPTOR
        (PACK_NAME : in STRING;
         IDENT     : in ACCESS_IDENTIFIER_DESCRIPTOR)
         return ACCESS_FULL_NAME_DESCRIPTOR;

function FIND_FULL_NAME_COMPONENT_DESCRIPTOR
        (PACK_NAME  : in STRING;
         IDENT      : in ACCESS_IDENTIFIER_DESCRIPTOR;
         TABLE_NAME : in STRING)
         return ACCESS_FULL_NAME_DESCRIPTOR;

function GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR
        (IDENT_DES          : in ACCESS_IDENTIFIER_DESCRIPTOR;
         TRY_OUTTER         : in STRING;
         TRY_OUTTER_LAST    : in NATURAL;
         TRY_INNER          : in STRING;
         TRY_INNER_LAST     : in NATURAL;
         KNOWN_OUTTER       : in STRING;
         KNOWN_OUTTER_LAST  : in NATURAL;
         KNOWN_INNER        : in STRING;
         KNOWN_INNER_LAST   : in NATURAL)
         return ACCESS_FULL_NAME_DESCRIPTOR;

function FIND_FULL_NAME_DESCRIPTOR_VISIBLE
        (SCHEMA         : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
         IDENT_DES      : in ACCESS_IDENTIFIER_DESCRIPTOR;
         OUTTER_PACKAGE : in STRING;
         OUTTER_LAST    : in NATURAL;
         INNER_PACKAGE  : in STRING;
         INNER_LAST     : in NATURAL)
         return ACCESS_FULL_NAME_DESCRIPTOR;

procedure BASE_TYPE_INTEGER
         (FULL_DES    : in ACCESS_FULL_NAME_DESCRIPTOR;
          IS_INT      : out BOOLEAN;
          LO_RANGE    : out INT;
          HI_RANGE    : out INT);

procedure LOCATE_PREVIOUS_IDENTIFIER
        (FULL_IDENT      : in out STRING;
         FULL_IDENT_LAST : in out NATURAL;
         IDENT_DES       : in out ACCESS_IDENTIFIER_DESCRIPTOR;
         FULL_DES        : in out ACCESS_FULL_NAME_DESCRIPTOR;
         ERROR           : in out INTEGER;
         ERR_MSG         : in BOOLEAN);

procedure STRING_TO_INT
```

```
        (INT_STRING : in STRING;
         OK         : out BOOLEAN;
         OUT_INT    : out INT);

  function BASE_TYPE_CHAR
        (FULL_DES : in ACCESS_FULL_NAME_DESCRIPTOR)
         return BOOLEAN;

  procedure IS_IDENTIFIER_NULL_OR_UNIQUE
        (THING     : in STRING;
         IS_NULL   : out BOOLEAN;
         IS_UNIQUE : out BOOLEAN);

  function IN_ADA_SQL_PACKAGE
         return BOOLEAN;

  procedure ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
         (IDENT_DES         : in out ACCESS_IDENTIFIER_DESCRIPTOR;
          FULL_DES          : in out ACCESS_FULL_NAME_DESCRIPTOR;
          NAME              : in STRING);

  procedure ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS
         (IDENT_DES         : in out ACCESS_IDENTIFIER_DESCRIPTOR;
          FULL_DES          : in out ACCESS_FULL_NAME_DESCRIPTOR;
          NAME              : in STRING;
          TABLE_NAME        : in STRING);

end SUBROUTINES_2_ROUTINES;
```

3.11.103 package ddl_subroutines_4_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS,
     SCHEMA_IO, DDL_VARIABLES, GET_NEW_DESCRIPTOR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES;
use  IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS,
     SCHEMA_IO, DDL_VARIABLES, GET_NEW_DESCRIPTOR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES;

package SUBROUTINES_4_ROUTINES is

  procedure WITH_USE_SCHEMA_DEFINITION
         (SCHEMA_DEF : in out BOOLEAN;
          OTHERS_TOO : in out BOOLEAN);

  procedure IS_AUTH_ID_UNIQUE
         (AUTH_ID    : in STRING;
          IS_UNIQUE  : in out BOOLEAN);
```

```
  procedure VALIDATE_NULL_UNIQUE_CONSTRAINTS
           (SUBTYPE_DES : in ACCESS_TYPE_DESCRIPTOR;
            PARENT_DES  : in ACCESS_TYPE_DESCRIPTOR;
            NULL_UNIQUE : in out BOOLEAN;
            VALID       : in out BOOLEAN);

  function NULL_UNIQUE_NAMES_THE_SAME
          (SUBTYPE_NAME     : in STRING;
           SUBTYPE_NULL     : in BOOLEAN;
           SUBTYPE_UNIQUE   : in BOOLEAN;
           PARENT_NAME      : in STRING;
           PARENT_NULL      : in BOOLEAN;
           PARENT_UNIQUE    : in BOOLEAN)
           return BOOLEAN;

  procedure SET_UP_WITH_USE_STANDARD_FOR_SCHEMA
           (THIS_SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR);

  procedure ADD_NEW_ENUM_LIT
           (TYPE_DES  : in ACCESS_TYPE_DESCRIPTOR;
            NAME      : in STRING);

  function FIND_EXISTING_ENUM_LIT
          (ENUM_LIT : in STRING)
           return ACCESS_ENUM_LIT_DESCRIPTOR;

  procedure ADD_NEW_ENUM_LIT_FOR_DERIVED
           (DERIVED_DES : in ACCESS_TYPE_DESCRIPTOR);

end SUBROUTINES_4_ROUTINES;
```

3.11.104 package ddl_subroutines_4.ada

```
package body SUBROUTINES_4_ROUTINES is

------------------------------------------------------------------------------
--
-- WITH_USE_SCHEMA_DEFINITION
--
-- tell me if we've withed and used schema definitions and if any other
-- packagew were withed and/or used, not counting ddl_standard_for_ada_sql

  procedure WITH_USE_SCHEMA_DEFINITION
           (SCHEMA_DEF : in out BOOLEAN;
            OTHERS_TOO : in out BOOLEAN) is

    WITHED : ACCESS_WITHED_UNIT_DESCRIPTOR  := CURRENT_SCHEMA_UNIT.FIRST_WITHED;
    USED   : ACCESS_USED_PACKAGE_DESCRIPTOR := CURRENT_SCHEMA_UNIT.FIRST_USED;
    WITHED_SCHEMA_DEF : BOOLEAN := FALSE;
    USED_SCHEMA_DEF   : BOOLEAN := FALSE;
```

```
    WITHED_OTHERS       : BOOLEAN := FALSE;
    USED_OTHERS         : BOOLEAN := FALSE;

  begin
    SCHEMA_DEF := FALSE;
    OTHERS_TOO  := FALSE;
    while WITHED /= null loop
      if CHARACTER_STRINGS_MATCH (STRING (WITHED.SCHEMA_UNIT.NAME.all),
                                  SCHEMA_DEF_NAME)
         then
        WITHED_SCHEMA_DEF := TRUE;
      elsif not CHARACTER_STRINGS_MATCH (STRING (WITHED.SCHEMA_UNIT.NAME.all),
                STANDARD_NAME) then
        WITHED_OTHERS := TRUE;
      end if;
      WITHED := WITHED.NEXT_WITHED;
    end loop;
    while USED /= null loop
      if CHARACTER_STRINGS_MATCH (STRING (USED.NAME.all), SCHEMA_DEF_NAME) then
        USED_SCHEMA_DEF := TRUE;
      elsif STRING (USED.NAME.all) /= STANDARD_NAME and
            STRING (USED.NAME.all) /= STANDARD_NAME_ADA_SQL then
        USED_OTHERS := TRUE;
      end if;
      USED := USED.NEXT_USED;
    end loop;
    if WITHED_SCHEMA_DEF and USED_SCHEMA_DEF then
      SCHEMA_DEF := TRUE;
    end if;
    if WITHED_OTHERS or USED_OTHERS then
      OTHERS_TOO := TRUE;
    end if;
  end WITH_USE_SCHEMA_DEFINITION;

------------------------------------------------------------------------------
--
-- IS_AUTH_ID_UNIQUE
--
-- return true if it is and false if it's not. Also print error message.

  procedure IS_AUTH_ID_UNIQUE
           (AUTH_ID    : in STRING;
            IS_UNIQUE  : in out BOOLEAN) is

    TEST_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR := FIRST_SCHEMA_UNIT;
    COUNT       : INTEGER := 0;

  begin
    IS_UNIQUE := TRUE;
```

```
    while TEST_SCHEMA /= null loop
      if TEST_SCHEMA /= CURRENT_SCHEMA_UNIT and then
         TEST_SCHEMA.IS_AUTH_PACKAGE and then
         STRING (TEST_SCHEMA.AUTH_ID.all) = AUTH_ID then
        if COUNT = 0 then
          PRINT_ERROR ("Duplicate authorization identifier: " & AUTH_ID);
          PRINT_TO_FILE ("  also declared in schema: " &
                         STRING (TEST_SCHEMA.NAME.all));
        else
          PRINT_TO_FILE ("  also declared in schema: " &
                         STRING (TEST_SCHEMA.NAME.all));
        end if;
        IS_UNIQUE := FALSE;
        COUNT := COUNT + 1;
      end if;
      TEST_SCHEMA := TEST_SCHEMA.NEXT_SCHEMA_UNIT;
    end loop;
  end IS_AUTH_ID_UNIQUE;

--------------------------------------------------------------------------------
--
-- VALIDATE_NULL_UNIQUE_CONSTRAINTS
--
-- given a subtype descriptor, whose NOT_NULL and NOT_UNIQUE variables reflect
-- the parents, determine if the subtype is more constrained than the parent.
-- also if constraints are involved then the basic name, without suffixes,
-- must be the same.

  procedure VALIDATE_NULL_UNIQUE_CONSTRAINTS
           (SUBTYPE_DES : in ACCESS_TYPE_DESCRIPTOR;
            PARENT_DES  : in ACCESS_TYPE_DESCRIPTOR;
            NULL_UNIQUE : in out BOOLEAN;
            VALID       : in out BOOLEAN) is

    IS_NULL     : BOOLEAN := FALSE;
    IS_UNIQUE   : BOOLEAN := FALSE;

  begin
    VALID := FALSE;
    NULL_UNIQUE := FALSE;
    IS_IDENTIFIER_NULL_OR_UNIQUE (STRING (SUBTYPE_DES.FULL_NAME.NAME.all),
               IS_NULL, IS_UNIQUE);
    if IS_NULL or IS_UNIQUE then
      NULL_UNIQUE := TRUE;
    end if;
    if not IS_NULL and not IS_UNIQUE and not PARENT_DES.NOT_NULL and
           not PARENT_DES.NOT_NULL_UNIQUE then
      VALID := TRUE;
      return;
```

```
    end if;
    if not NULL_UNIQUE_NAMES_THE_SAME (STRING (SUBTYPE_DES.FULL_NAME.NAME.all),
               IS_NULL, IS_UNIQUE, STRING (PARENT_DES.FULL_NAME.NAME.all),
               PARENT_DES.NOT_NULL, PARENT_DES.NOT_NULL_UNIQUE) then
      PRINT_ERROR ("Identifier: " & STRING (SUBTYPE_DES.FULL_NAME.NAME.all) &
                   "cannot be");
      PRINT_TO_FILE ("   a constrained subtype of identifier: " &
                     STRING (PARENT_DES.FULL_NAME.NAME.all));
      return;
    end if;
    if ((IS_NULL or IS_UNIQUE) and
         not PARENT_DES.NOT_NULL and
         not PARENT_DES.NOT_NULL_UNIQUE) or
       (IS_NULL and
         not PARENT_DES.NOT_NULL and
         not PARENT_DES.NOT_NULL_UNIQUE) or
       (IS_UNIQUE and
         not PARENT_DES.NOT_NULL_UNIQUE) then
      VALID := TRUE;
      return;
    end if;
    PRINT_ERROR ("Subtype identifier: " &
                 STRING (SUBTYPE_DES.FULL_NAME.NAME.all));
    PRINT_TO_FILE ("  is less constrained than parent identifier: " &
                   STRING (PARENT_DES.FULL_NAME.NAME.all));
  end VALIDATE_NULL_UNIQUE_CONSTRAINTS;

--------------------------------------------------------------------------------
--
-- NULL_UNIQUE_NAMES_THE_SAME
--
-- lop off the suffixes and are the identifiers the same

  function NULL_UNIQUE_NAMES_THE_SAME
          (SUBTYPE_NAME    : in STRING;
           SUBTYPE_NULL    : in BOOLEAN;
           SUBTYPE_UNIQUE  : in BOOLEAN;
           PARENT_NAME     : in STRING;
           PARENT_NULL     : in BOOLEAN;
           PARENT_UNIQUE   : in BOOLEAN)
           return BOOLEAN is

    SUBTYPE_END : INTEGER := 0;
    PARENT_END  : INTEGER := 0;

  begin
    SUBTYPE_END := SUBTYPE_NAME'LAST;
    PARENT_END  := PARENT_NAME'LAST;
    if SUBTYPE_NULL then
```

```
      SUBTYPE_END := SUBTYPE_END - 9;
    end if;
    if SUBTYPE_UNIQUE then
      SUBTYPE_END := SUBTYPE_END - 16;
    end if;
    if PARENT_NULL then
      PARENT_END := PARENT_END - 9;
    end if;
    if PARENT_UNIQUE then
      PARENT_END := PARENT_END - 16;
    end if;
    if SUBTYPE_END < SUBTYPE_NAME'FIRST or
       PARENT_END  < PARENT_NAME'FIRST or
       SUBTYPE_NAME (SUBTYPE_NAME'FIRST..SUBTYPE_END) /=
       PARENT_NAME (PARENT_NAME'FIRST..PARENT_END) then
      return FALSE;
    else
      return TRUE;
    end if;
  end NULL_UNIQUE_NAMES_THE_SAME;

----------------------------------------------------------------------
--
-- SET_UP_WITH_USE_STANDARD_FOR_SCHEMA
--
-- if this schema is "DDL_STANDARD_FOR_ADA_SQL" then don't do anything
-- if we haven't already withed "DDL_STANDARD_FOR_ADA_SQL" then with it
-- if we haven't already used "DDL_STANDARD_FOR_ADA_SQL" and
-- "DDL_STANDARD_FOR_ADA_SQL.ADA_SQL" then use them

  procedure SET_UP_WITH_USE_STANDARD_FOR_SCHEMA
           (THIS_SCHEMA : in out ACCESS_SCHEMA_UNIT_DESCRIPTOR) is

    WITHED_UNIT_DES    : ACCESS_WITHED_UNIT_DESCRIPTOR := null;
    WITHED_UNIT_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
    USED_PACKAGE       : ACCESS_USED_PACKAGE_DESCRIPTOR := null;

  begin
    if CHARACTER_STRINGS_MATCH (STRING (THIS_SCHEMA.NAME.all),
                                STANDARD_NAME) then
      return;
    end if;
    WITHED_UNIT_SCHEMA := FIND_SCHEMA_UNIT_DESCRIPTOR (STANDARD_NAME);
    if not DUPLICATE_WITH (THIS_SCHEMA, WITHED_UNIT_SCHEMA) then
      WITHED_UNIT_DES := GET_NEW_WITHED_UNIT_DESCRIPTOR;
      WITHED_UNIT_DES.SCHEMA_UNIT := WITHED_UNIT_SCHEMA;
      ADD_WITHED_UNIT_DESCRIPTOR (WITHED_UNIT_DES, THIS_SCHEMA);
    end if;
    if not DUPLICATE_USE (THIS_SCHEMA, STANDARD_NAME) then
```

```
      USED_PACKAGE := GET_NEW_USED_PACKAGE_DESCRIPTOR;
      USED_PACKAGE.NAME := GET_NEW_PACKAGE_NAME (STANDARD_NAME);
      ADD_USED_PACKAGE_DESCRIPTOR (USED_PACKAGE, THIS_SCHEMA);
    end if;
    if not DUPLICATE_USE (THIS_SCHEMA, STANDARD_NAME_ADA_SQL) then
      USED_PACKAGE := GET_NEW_USED_PACKAGE_DESCRIPTOR;
      USED_PACKAGE.NAME := GET_NEW_PACKAGE_NAME (STANDARD_NAME_ADA_SQL);
      ADD_USED_PACKAGE_DESCRIPTOR (USED_PACKAGE, THIS_SCHEMA);
    end if;
  end SET_UP_WITH_USE_STANDARD_FOR_SCHEMA;

------------------------------------------------------------------------
--
-- ADD_NEW_ENUM_LIT
--
-- the enumeration literal descriptor may already exist, if not create one
-- the full enumeration literal des will not already exist, create it

  procedure ADD_NEW_ENUM_LIT
          (TYPE_DES   : in ACCESS_TYPE_DESCRIPTOR;
           NAME       : in STRING) is

    ENUM_DES : ACCESS_ENUM_LIT_DESCRIPTOR := null;
    FULL_DES : ACCESS_FULL_ENUM_LIT_DESCRIPTOR := null;

  begin
    ENUM_DES := FIND_EXISTING_ENUM_LIT (NAME);
    if ENUM_DES = null then
      ENUM_DES := GET_NEW_ENUM_LIT_DESCRIPTOR;
      ENUM_DES.NAME := GET_NEW_ENUM_LIT_NAME (NAME);
      ADD_ENUM_LIT_DESCRIPTOR (ENUM_DES);
    end if;
    FULL_DES := GET_NEW_FULL_ENUM_LIT_DESCRIPTOR;
    FULL_DES.NAME := GET_NEW_ENUM_LIT_NAME (NAME);
    FULL_DES.TYPE_IS := TYPE_DES;
    ADD_FULL_ENUM_LIT_DESCRIPTOR (FULL_DES, ENUM_DES);
  end ADD_NEW_ENUM_LIT;

----------------------------------------------------------------
--
-- FIND_EXISTING_ENUM_LIT
--
-- given an enumeration literal return it's enum lit descriptor

  function FIND_EXISTING_ENUM_LIT
         (ENUM_LIT : in STRING)
          return ACCESS_ENUM_LIT_DESCRIPTOR is

    LIT : ACCESS_ENUM_LIT_DESCRIPTOR := FIRST_ENUM_LIT;
```

```
  begin
    while LIT /= null loop
      exit when STRING (LIT.NAME.all) = ENUM_LIT;
      LIT := LIT.NEXT_ENUM_LIT;
    end loop;
    return LIT;
  end FIND_EXISTING_ENUM_LIT;

----------------------------------------------------------------------------
--
-- ADD_NEW_ENUM_LIT_FOR_DERIVED

  procedure ADD_NEW_ENUM_LIT_FOR_DERIVED
           (DERIVED_DES : in ACCESS_TYPE_DESCRIPTOR) is

    LIT : ACCESS_LITERAL_DESCRIPTOR := DERIVED_DES.FIRST_LITERAL;

  begin
    while LIT /= null loop
      ADD_NEW_ENUM_LIT (DERIVED_DES, STRING (LIT.NAME.all));
      if LIT = DERIVED_DES.LAST_LITERAL then
        LIT := null;
      else
        LIT := LIT.NEXT_LITERAL;
      end if;
    end loop;
  end ADD_NEW_ENUM_LIT_FOR_DERIVED;

end SUBROUTINES_4_ROUTINES;
```

3.11.105 package ddl_subroutines_3_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES;

package SUBROUTINES_3_ROUTINES is

  procedure BREAK_DOWN_SUBTYPE_INDICATOR
           (VALID               : in out BOOLEAN;
            ERROR_NUMBER        : in out NATURAL;
            TYPE_DES            : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in out BOOLEAN;
            ARRAY_INDEX_LO      : in out INT;
            ARRAY_INDEX_HI      : in out INT;
            GOT_INTEGER_RANGE   : in out BOOLEAN;
            INTEGER_RANGE_LO    : in out INT;
            INTEGER_RANGE_HI    : in out INT;
            GOT_FLOAT_DIGITS    : in out BOOLEAN;
```

```
             FLOAT_DIGITS         : in out NATURAL;
             GOT_FLOAT_RANGE      : in out BOOLEAN;
             FLOAT_RANGE_LO       : in out DOUBLE_PRECISION;
             FLOAT_RANGE_HI       : in out DOUBLE_PRECISION;
             GOT_ENUM_RANGE       : in out BOOLEAN;
             ENUM_RANGE_LO        : in out ACCESS_LITERAL_DESCRIPTOR;
             ENUM_RANGE_HI        : in out ACCESS_LITERAL_DESCRIPTOR;
             ENUM_POS             : in out NATURAL);

   procedure SUBTYPE_INDICATOR_IS_ENUMERATION
            (VALID                : in out BOOLEAN;
             ERROR_NUMBER         : in out NATURAL;
             TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
             GOT_ENUM_RANGE       : in out BOOLEAN;
             ENUM_RANGE_LO        : in out ACCESS_LITERAL_DESCRIPTOR;
             ENUM_RANGE_HI        : in out ACCESS_LITERAL_DESCRIPTOR;
             ENUM_POS             : in out NATURAL);

   procedure LOCATE_ENUMERATION_LITERAL
           (TYPE_DES   : in ACCESS_TYPE_DESCRIPTOR;
            LIT        : in STRING;
            POS        : out NATURAL;
            LIT_DES    : out ACCESS_LITERAL_DESCRIPTOR);

   procedure SUBTYPE_INDICATOR_IS_INTEGER
            (VALID                : in out BOOLEAN;
             ERROR_NUMBER         : in out NATURAL;
             TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
             GOT_INTEGER_RANGE    : in out BOOLEAN;
             INTEGER_RANGE_LO     : in out INT;
             INTEGER_RANGE_HI     : in out INT);

   procedure SUBTYPE_INDICATOR_IS_FLOAT
            (VALID                : in out BOOLEAN;
             ERROR_NUMBER         : in out NATURAL;
             TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
             GOT_FLOAT_DIGITS     : out BOOLEAN;
             FLOAT_DIGITS         : out NATURAL;
             GOT_FLOAT_RANGE      : out BOOLEAN;
             FLOAT_RANGE_LO       : out DOUBLE_PRECISION;
             FLOAT_RANGE_HI       : out DOUBLE_PRECISION);

   procedure SUBTYPE_INDICATOR_IS_STRING
            (VALID                : in out BOOLEAN;
             ERROR_NUMBER         : in out NATURAL;
             TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
             GOT_ARRAY_INDEX      : in out BOOLEAN;
             ARRAY_INDEX_LO       : in out INT;
             ARRAY_INDEX_HI       : in out INT);
```

```
  procedure INSERT_SUBTYPE_INDICATOR_INFORMATION
           (PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
            NEW_DES             : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in BOOLEAN;
            ARRAY_INDEX_LO      : in INT;
            ARRAY_INDEX_HI      : in INT;
            GOT_INTEGER_RANGE   : in BOOLEAN;
            INTEGER_RANGE_LO    : in INT;
            INTEGER_RANGE_HI    : in INT;
            GOT_FLOAT_DIGITS    : in BOOLEAN;
            FLOAT_DIGITS        : in NATURAL;
            GOT_FLOAT_RANGE     : in BOOLEAN;
            FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE      : in BOOLEAN;
            ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS            : in NATURAL);

end SUBROUTINES_3_ROUTINES;
```

3.11.106 package ddl_subroutines_3.ada

```
package body SUBROUTINES_3_ROUTINES is

-----------------------------------------------------------------------
--
-- BREAK_DOWN_SUBTYPE_INDICATOR
--
-- on entry temp_string should contain the previous identifier of the
-- subtype indicator.  If that type is:
--   unconstrained array - may or may not specify a range and we will return
--                      got_array_index, array_index_lo and array_index_hi
--   constrained array - must specify nothing else
--   integer - may specify a range, return got_integer_range, integer_ragne_lo
--             and integer_range_hi
--   real - may specify a digits and or a range, return got_float_digits,
--          float_digits, got_float_range, float_range_lo and float_range_hi
--   enumeration - may specify a range, return got_enum_range, enum_range_lo,
--                 and enum_range_hi
--   record - invalid
--
--  errors returned:
--    1   the previous identifier was invalid
--    2   the previous identifier was a component
--    3   the previous identifier was a record
--    4   for enumeration range not found but something bogus there
--    5   for enumeration range literals are incorrect
--    6   for integer range not found but something bogus there
--    7   for integer range integersare incorrect
```

```
--    8   for float expecting digits or range or ; found none
--    9   for float digits integers are incorrect
--   10   for float range integers are incorrect
--   11   for string range not found but something bogus there
--   12   for string range is incorrect
--   13   for string range was given for a constrained array
--   14   no longer used - for string range was not given for an
--        unconstrained array
--

  procedure BREAK_DOWN_SUBTYPE_INDICATOR
           (VALID              : in out BOOLEAN;
            ERROR_NUMBER       : in out NATURAL;
            TYPE_DES           : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX    : in out BOOLEAN;
            ARRAY_INDEX_LO     : in out INT;
            ARRAY_INDEX_HI     : in out INT;
            GOT_INTEGER_RANGE  : in out BOOLEAN;
            INTEGER_RANGE_LO   : in out INT;
            INTEGER_RANGE_HI   : in out INT;
            GOT_FLOAT_DIGITS   : in out BOOLEAN;
            FLOAT_DIGITS       : in out NATURAL;
            GOT_FLOAT_RANGE    : in out BOOLEAN;
            FLOAT_RANGE_LO     : in out DOUBLE_PRECISION;
            FLOAT_RANGE_HI     : in out DOUBLE_PRECISION;
            GOT_ENUM_RANGE     : in out BOOLEAN;
            ENUM_RANGE_LO      : in out ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI      : in out ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS           : in out NATURAL) is

    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    ERROR       : INTEGER := 0;

  begin
    VALID                := TRUE;
    ERROR_NUMBER         := 0;
    TYPE_DES             := null;
    GOT_ARRAY_INDEX      := FALSE;
    ARRAY_INDEX_LO       := 0;
    ARRAY_INDEX_HI       := 0;
    GOT_INTEGER_RANGE    := FALSE;
    INTEGER_RANGE_LO     := 0;
    INTEGER_RANGE_HI     := 0;
    GOT_FLOAT_DIGITS     := FALSE;
    FLOAT_DIGITS         := 0;
    GOT_FLOAT_RANGE      := FALSE;
    FLOAT_RANGE_LO       := 0.0;
    FLOAT_RANGE_HI       := 0.0;
```

```
    GOT_ENUM_RANGE      :=  FALSE;
    ENUM_RANGE_LO       :=  null;
    ENUM_RANGE_HI       :=  null;
    ENUM_POS            :=  0;

    LOCATE_PREVIOUS_IDENTIFIER (TEMP_STRING (1..TEMP_STRING_LAST),
                    TEMP_STRING_LAST, IDENT_DES, FULL_DES, ERROR, FALSE);
    if ERROR /= 0 then
      VALID := FALSE;
      ERROR_NUMBER := 1;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    TYPE_DES := FULL_DES.TYPE_IS;
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if TYPE_DES.TYPE_KIND /= A_COMPONENT then
      case TYPE_DES.WHICH_TYPE is
        when REC_ORD      => VALID := FALSE;
                              ERROR_NUMBER := 3;
                              FIND_END_OF_STATEMENT (TEMP_STRING,
                                                     TEMP_STRING_LAST);
                              return;
        when ENUMERATION => SUBTYPE_INDICATOR_IS_ENUMERATION (VALID,
                             ERROR_NUMBER, TYPE_DES, GOT_ENUM_RANGE,
                             ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
        when INT_EGER    => SUBTYPE_INDICATOR_IS_INTEGER (VALID,
                             ERROR_NUMBER, TYPE_DES, GOT_INTEGER_RANGE,
                             INTEGER_RANGE_LO, INTEGER_RANGE_HI);
        when FL_OAT      => SUBTYPE_INDICATOR_IS_FLOAT (VALID,
                             ERROR_NUMBER, TYPE_DES, GOT_FLOAT_DIGITS,
                             FLOAT_DIGITS, GOT_FLOAT_RANGE, FLOAT_RANGE_LO,
                             FLOAT_RANGE_HI);
        when STR_ING     => SUBTYPE_INDICATOR_IS_STRING (VALID,
                             ERROR_NUMBER, TYPE_DES, GOT_ARRAY_INDEX,
                             ARRAY_INDEX_LO, ARRAY_INDEX_HI);
      end case;
    else
      VALID := FALSE;
      ERROR_NUMBER := 2;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
  end BREAK_DOWN_SUBTYPE_INDICATOR;

------------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_ENUMERATION
--
-- on entry temp_string should contain either ; or RANGE
```

```
-- if ; then just return valid=true
-- if range then it must be followed by two enumeration literal range
-- specifiers.  They must be located in the parent (type_des) and ordered
-- correctly, if so return them, if not error
--
--   errors returned:
--     4    range not found but something bogus there
--     5    range literals are incorrect
--

  procedure SUBTYPE_INDICATOR_IS_ENUMERATION
          (VALID                  : in out BOOLEAN;
           ERROR_NUMBER           : in out NATURAL;
           TYPE_DES               : in out ACCESS_TYPE_DESCRIPTOR;
           GOT_ENUM_RANGE         : in out BOOLEAN;
           ENUM_RANGE_LO          : in out ACCESS_LITERAL_DESCRIPTOR;
           ENUM_RANGE_HI          : in out ACCESS_LITERAL_DESCRIPTOR;
           ENUM_POS               : in out NATURAL) is

    OK        : BOOLEAN := TRUE;
    LIT1      : STRING (1..25) := (others => ' ');
    LIT1_LAST : NATURAL := 0;
    LIT2      : STRING (1..25) := (others => ' ');
    LIT2_LAST : NATURAL := 0;
    POS1      : NATURAL := 0;
    POS2      : NATURAL := 0;
    LIT1_DES  : ACCESS_LITERAL_DESCRIPTOR := null;
    LIT2_DES  : ACCESS_LITERAL_DESCRIPTOR := null;

  begin

    VALID              :=  TRUE;
    ERROR_NUMBER       :=  0;
    GOT_ENUM_RANGE     :=  FALSE;
    ENUM_RANGE_LO      :=  null;
    ENUM_RANGE_HI      :=  null;
    ENUM_POS           :=  0;

 -- first we either have ; or RANGE

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) /= "RANGE" then
      VALID := FALSE;
      ERROR_NUMBER := 4;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    GOT_ENUM_RANGE := TRUE;
```

```
-- now find first range literal

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      return;
    end if;
    LIT1_LAST := TEMP_STRING_LAST;
    LIT1 (1..LIT1_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
    OK := VALID;
    if TEMP_STRING (1..TEMP_STRING_LAST) = "'" then
      GET_SINGLE_QUOTE_STRING (CURRENT_SCHEMA_UNIT, LIT1, LIT1_LAST, OK);
    end if;
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find .. between literals

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_CONSTANT (OK, ".", TRUE);
    GET_CONSTANT (OK, ".", TRUE);
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find range literal 2

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      return;
    end if;
    LIT2_LAST := TEMP_STRING_LAST;
    LIT2 (1..LIT2_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
    OK := VALID;
    if TEMP_STRING (1..TEMP_STRING_LAST) = "'" then
      GET_SINGLE_QUOTE_STRING (CURRENT_SCHEMA_UNIT, LIT2, LIT2_LAST, OK);
    end if;
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 5;
```

```
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now we should be at the end of the statement

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find out if the literals belong to the parents

    LOCATE_ENUMERATION_LITERAL (TYPE_DES, LIT1 (1..LIT1_LAST), POS1, LIT1_DES);
    LOCATE_ENUMERATION_LITERAL (TYPE_DES, LIT2 (1..LIT2_LAST), POS2, LIT2_DES);
    if POS1 = 0 or POS2 = 0 or POS1 > POS2 then
      VALID := FALSE;
      ERROR_NUMBER := 5;
      return;
    end if;
    ENUM_POS := POS2 - POS1 + 1;
    ENUM_RANGE_LO := LIT1_DES;
    ENUM_RANGE_HI := LIT2_DES;
  end SUBTYPE_INDICATOR_IS_ENUMERATION;

----------------------------------------------------------------------
--
-- LOCATE_ENUMERATION_LITERAL
--
-- return the position and descriptor of the given literal if it appears
-- in the given type descriptor

  procedure LOCATE_ENUMERATION_LITERAL
          (TYPE_DES  : in ACCESS_TYPE_DESCRIPTOR;
           LIT       : in STRING;
           POS       : out NATURAL;
           LIT_DES   : out ACCESS_LITERAL_DESCRIPTOR) is

    CHECK_LIT : ACCESS_LITERAL_DESCRIPTOR := TYPE_DES.FIRST_LITERAL;

  begin
    POS := 0;
    LIT_DES := null;
    while CHECK_LIT /= null loop
      if STRING (CHECK_LIT.NAME.all) = LIT then
        LIT_DES := CHECK_LIT;
```

```
        POS := CHECK_LIT.POS;
        return;
      end if;
      exit when CHECK_LIT = TYPE_DES.LAST_LITERAL;
      CHECK_LIT := CHECK_LIT.NEXT_LITERAL;
    end loop;
  end LOCATE_ENUMERATION_LITERAL;

-----------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_INTEGER
--
-- on entry temp_string should contain either ; or RANGE
-- if ; then just return valid=true
-- if range then it must be followed by two integer range
-- specifiers.  They must fuall within the range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
--    6    range not found but something bogus there
--    7    range integers are incorrect
--

  procedure SUBTYPE_INDICATOR_IS_INTEGER
           (VALID                : in out BOOLEAN;
            ERROR_NUMBER         : in out NATURAL;
            TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_INTEGER_RANGE    : in out BOOLEAN;
            INTEGER_RANGE_LO     : in out INT;
            INTEGER_RANGE_HI     : in out INT) is

    OK          : BOOLEAN := TRUE;
    RANGE1      : INT := 0;
    RANGE2      : INT := 0;

  begin

    VALID               :=  TRUE;
    ERROR_NUMBER        :=  0;
    GOT_INTEGER_RANGE   :=  FALSE;
    INTEGER_RANGE_LO    :=  0;
    INTEGER_RANGE_HI    :=  0;

-- first we either have ; or RANGE

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) /= "RANGE" then
      VALID := FALSE;
```

```
        ERROR_NUMBER := 6;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
      GOT_INTEGER_RANGE := TRUE;

  -- now find lo range

      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        VALID := FALSE;
        ERROR_NUMBER := 7;
        return;
      end if;
      OK := TRUE;
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE1);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 7;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;

  -- now find .. between integers

      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      GET_CONSTANT (OK, ".", TRUE);
      GET_CONSTANT (OK, ".", TRUE);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 7;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;

  -- now find hi range

      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        VALID := FALSE;
        ERROR_NUMBER := 7;
        return;
      end if;
      OK := VALID;
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE2);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 7;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
```

```
    end if;

-- now we should be at the end of the statement

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 7;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find out if the range is valid with the parents

    if RANGE1 > RANGE2 or RANGE1 < TYPE_DES.RANGE_LO_INT or
       RANGE2 > TYPE_DES.RANGE_HI_INT then
      VALID := FALSE;
      ERROR_NUMBER := 7;
      return;
    else
      INTEGER_RANGE_LO := RANGE1;
      INTEGER_RANGE_HI := RANGE2;
    end if;
  end SUBTYPE_INDICATOR_IS_INTEGER;

-----------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_FLOAT
--
-- on entry temp_string should contain either ; or DIGITS or RANGE
-- if ; then just return valid=true
-- if digits then it must be followed by an integer
-- if range then it must be followed by two floats
-- They must fall within the digits and range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
--     8    expecting digits or range or ; found none
--     9    digits is incorrect
--    10    range is incorrect

  procedure SUBTYPE_INDICATOR_IS_FLOAT
           (VALID              : in out BOOLEAN;
            ERROR_NUMBER       : in out NATURAL;
            TYPE_DES           : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_FLOAT_DIGITS   : out BOOLEAN;
            FLOAT_DIGITS       : out NATURAL;
            GOT_FLOAT_RANGE    : out BOOLEAN;
            FLOAT_RANGE_LO     : out DOUBLE_PRECISION;
```

```
            FLOAT_RANGE_HI     : out DOUBLE_PRECISION) is

    OK          : BOOLEAN := TRUE;
    RANGE1      : DOUBLE_PRECISION := 0.0;
    RANGE2      : DOUBLE_PRECISION := 0.0;
    DIGIT_INT : INT := 0;

  begin

    VALID               :=  TRUE;
    ERROR_NUMBER        :=  0;
    GOT_FLOAT_DIGITS    :=  FALSE;
    FLOAT_DIGITS        :=  0;
    GOT_FLOAT_RANGE     :=  FALSE;
    FLOAT_RANGE_LO      :=  0.0;
    FLOAT_RANGE_HI      :=  0.0;

-- first we either have ; or DIGITS or RANGE

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) /= "DIGITS" and
          TEMP_STRING (1..TEMP_STRING_LAST) /= "RANGE" then
      VALID := FALSE;
      ERROR_NUMBER := 8;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- process DIGITS here

    if TEMP_STRING (1..TEMP_STRING_LAST) = "DIGITS" then
      GOT_FLOAT_DIGITS := TRUE;
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      OK := VALID;
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, DIGIT_INT);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 9;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
      if (DIGIT_INT < 1) or (DIGIT_INT > INT (TYPE_DES.FLOAT_DIGITS)) then
        VALID := FALSE;
        ERROR_NUMBER := 9;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
      FLOAT_DIGITS := INTEGER (DIGIT_INT);
```

```
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    end if;

-- process range here

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) /= "RANGE" then
      VALID := FALSE;
      ERROR_NUMBER := 8;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

    GOT_FLOAT_RANGE := TRUE;

-- now find lo range

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 10;
      return;
    end if;
    OK := TRUE;
    STRING_TO_DOUBLE_PRECISION (TEMP_STRING (1..TEMP_STRING_LAST),
                                OK, RANGE1);
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 10;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find .. between floats

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_CONSTANT (OK, ".", TRUE);
    GET_CONSTANT (OK, ".", TRUE);
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 10;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find hi range

    if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
```

```
      VALID := FALSE;
      ERROR_NUMBER := 10;
      return;
    end if;
    OK := VALID;
    STRING_TO_DOUBLE_PRECISION (TEMP_STRING (1..TEMP_STRING_LAST),
                                OK, RANGE2);
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 10;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find out if the range is valid with the parents

    if RANGE1 > RANGE2 or RANGE1 < TYPE_DES.RANGE_LO_FLT or
       RANGE2 > TYPE_DES.RANGE_HI_FLT then
      VALID := FALSE;
      ERROR_NUMBER := 10;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    else
      FLOAT_RANGE_LO := RANGE1;
      FLOAT_RANGE_HI := RANGE2;
    end if;

-- now we should be at the end of the statement

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 8;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
  end SUBTYPE_INDICATOR_IS_FLOAT;

----------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_STRING
--
-- on entry temp_string should contain either ; or (
-- if ; then just return valid=true
-- if ( then it must be followed by a range and )
-- Range must fall within the range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
```

```
--     11    range not found but something bogus there
--     12    range is incorrect
--     13    range was given for a constrained array
--     14    no longer used - range was not given for an unconstrained array

  procedure SUBTYPE_INDICATOR_IS_STRING
           (VALID                : in out BOOLEAN;
            ERROR_NUMBER         : in out NATURAL;
            TYPE_DES             : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX      : in out BOOLEAN;
            ARRAY_INDEX_LO       : in out INT;
            ARRAY_INDEX_HI       : in out INT) is

    OK          : BOOLEAN := TRUE;
    RANGE1      : INT := 0;
    RANGE2      : INT := 0;

  begin

    VALID                :=  TRUE;
    ERROR_NUMBER         :=  0;
    GOT_ARRAY_INDEX      :=  FALSE;
    ARRAY_INDEX_LO       :=  0;
    ARRAY_INDEX_HI       :=  0;

-- first we either have ; or (
-- if constrained parent and range supplied = error
-- if unconstrained parent may or may not have range

    if TYPE_DES.CONSTRAINED then
      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        return;
      else
        VALID := FALSE;
        ERROR_NUMBER := 13;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
    else
      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
--          VALID := FALSE;
--          ERROR_NUMBER := 14;
        return;
      end if;
    end if;
    if TEMP_STRING (1..TEMP_STRING_LAST) /= "(" then
      VALID := FALSE;
      ERROR_NUMBER := 11;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
```

```
        return;
      end if;
      GOT_ARRAY_INDEX := TRUE;

  -- now find lo range

      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        VALID := FALSE;
        ERROR_NUMBER := 12;
        return;
      end if;
      OK := TRUE;
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE1);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 12;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;

  -- now find .. between integers

      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      GET_CONSTANT (OK, ".", TRUE);
      GET_CONSTANT (OK, ".", TRUE);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 12;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;

  -- now find hi range

      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        VALID := FALSE;
        ERROR_NUMBER := 12;
        return;
      end if;
      OK := VALID;
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE2);
      if not OK then
        VALID := FALSE;
        ERROR_NUMBER := 12;
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
```

```
-- now we should be at the end of the statement find );

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_CONSTANT (OK, ")", TRUE);
    if not OK then
      VALID := FALSE;
      ERROR_NUMBER := 12;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      VALID := FALSE;
      ERROR_NUMBER := 12;
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- now find out if the range is valid with the parents

    if RANGE1 > RANGE2 or
       ((TYPE_DES.ARRAY_RANGE_MIN /= -1 or TYPE_DES.ARRAY_RANGE_MAX /= -1)
        and then (RANGE1 < TYPE_DES.ARRAY_RANGE_MIN or
                  RANGE2 > TYPE_DES.ARRAY_RANGE_MAX)) then
      VALID := FALSE;
      ERROR_NUMBER := 12;
      return;
    else
      ARRAY_INDEX_LO := RANGE1;
      ARRAY_INDEX_HI := RANGE2;
    end if;
  end SUBTYPE_INDICATOR_IS_STRING;

------------------------------------------------------------------------
--
-- INSERT_SUBTYPE_INDICATOR_INFORMATION
--
  procedure INSERT_SUBTYPE_INDICATOR_INFORMATION
           (PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
            NEW_DES             : in out ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in BOOLEAN;
            ARRAY_INDEX_LO      : in INT;
            ARRAY_INDEX_HI      : in INT;
            GOT_INTEGER_RANGE   : in BOOLEAN;
            INTEGER_RANGE_LO    : in INT;
            INTEGER_RANGE_HI    : in INT;
            GOT_FLOAT_DIGITS    : in BOOLEAN;
            FLOAT_DIGITS        : in NATURAL;
            GOT_FLOAT_RANGE     : in BOOLEAN;
            FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
```

```
          FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
          GOT_ENUM_RANGE      : in BOOLEAN;
          ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
          ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
          ENUM_POS            : in NATURAL) is
begin
  NEW_DES.WHICH_TYPE  := PARENT_DES.WHICH_TYPE;
  NEW_DES.PARENT_TYPE := PARENT_DES;
  NEW_DES.NOT_NULL    := PARENT_DES.NOT_NULL;
  NEW_DES.NOT_NULL_UNIQUE := PARENT_DES.NOT_NULL_UNIQUE;
  case NEW_DES.TYPE_KIND is
    when A_DERIVED => NEW_DES.BASE_TYPE := NEW_DES;
                      NEW_DES.ULT_PARENT_TYPE := PARENT_DES.ULT_PARENT_TYPE;
    when others    => NEW_DES.BASE_TYPE := PARENT_DES.BASE_TYPE;
                      NEW_DES.ULT_PARENT_TYPE := PARENT_DES.ULT_PARENT_TYPE;
  end case;
  case PARENT_DES.WHICH_TYPE is
    when REC_ORD     => null;
    when ENUMERATION =>
         if GOT_ENUM_RANGE then
           NEW_DES.FIRST_LITERAL := ENUM_RANGE_LO;
           NEW_DES.LAST_LITERAL  := ENUM_RANGE_HI;
           NEW_DES.LAST_POS      := ENUM_POS;
           NEW_DES.MAX_LENGTH    := PARENT_DES.MAX_LENGTH;
         else
           NEW_DES.FIRST_LITERAL := PARENT_DES.FIRST_LITERAL;
           NEW_DES.LAST_LITERAL  := PARENT_DES.LAST_LITERAL;
           NEW_DES.LAST_POS      := PARENT_DES.LAST_POS;
           NEW_DES.MAX_LENGTH    := PARENT_DES.MAX_LENGTH;
         end if;
    when INT_EGER    =>
         if GOT_INTEGER_RANGE then
           NEW_DES.RANGE_LO_INT := INTEGER_RANGE_LO;
           NEW_DES.RANGE_HI_INT := INTEGER_RANGE_HI;
         else
           NEW_DES.RANGE_LO_INT := PARENT_DES.RANGE_LO_INT;
           NEW_DES.RANGE_HI_INT := PARENT_DES.RANGE_HI_INT;
         end if;
    when FL_OAT      =>
         if GOT_FLOAT_DIGITS then
           NEW_DES.FLOAT_DIGITS := FLOAT_DIGITS;
         else
           NEW_DES.FLOAT_DIGITS := PARENT_DES.FLOAT_DIGITS;
         end if;
         if GOT_FLOAT_RANGE then
           NEW_DES.RANGE_LO_FLT := FLOAT_RANGE_LO;
           NEW_DES.RANGE_HI_FLT := FLOAT_RANGE_HI;
         else
           NEW_DES.RANGE_LO_FLT := PARENT_DES.RANGE_LO_FLT;
```

```
              NEW_DES.RANGE_HI_FLT := PARENT_DES.RANGE_HI_FLT;
            end if;
        when STR_ING     =>
            NEW_DES.INDEX_TYPE := PARENT_DES.INDEX_TYPE;
            NEW_DES.ARRAY_TYPE := PARENT_DES.ARRAY_TYPE;
            NEW_DES.ARRAY_RANGE_MIN := PARENT_DES.ARRAY_RANGE_MIN;
            NEW_DES.ARRAY_RANGE_MAX := PARENT_DES.ARRAY_RANGE_MAX;
            if GOT_ARRAY_INDEX then
              NEW_DES.CONSTRAINED := TRUE;
              NEW_DES.ARRAY_RANGE_LO := ARRAY_INDEX_LO;
              NEW_DES.ARRAY_RANGE_HI := ARRAY_INDEX_HI;
              NEW_DES.LENGTH := INTEGER (ARRAY_INDEX_HI - ARRAY_INDEX_LO + 1);
            else
              NEW_DES.CONSTRAINED := PARENT_DES.CONSTRAINED;
              NEW_DES.ARRAY_RANGE_LO := PARENT_DES.ARRAY_RANGE_LO;
              NEW_DES.ARRAY_RANGE_HI := PARENT_DES.ARRAY_RANGE_HI;
              NEW_DES.LENGTH := PARENT_DES.LENGTH;
            end if;
      end case;
      case NEW_DES.TYPE_KIND is
        when A_SUBTYPE   => if PARENT_DES.FIRST_SUBTYPE = null then
                               PARENT_DES.FIRST_SUBTYPE := NEW_DES;
                             else
                               PARENT_DES.LAST_SUBTYPE.NEXT_ONE := NEW_DES;
                             end if;
                             NEW_DES.PREVIOUS_ONE := PARENT_DES.LAST_SUBTYPE;
                             PARENT_DES.LAST_SUBTYPE := NEW_DES;
        when A_TYPE      => null;
        when A_DERIVED   => if PARENT_DES.FIRST_DERIVED = null then
                               PARENT_DES.FIRST_DERIVED := NEW_DES;
                             else
                               PARENT_DES.LAST_DERIVED.NEXT_ONE := NEW_DES;
                             end if;
                             NEW_DES.PREVIOUS_ONE := PARENT_DES.LAST_DERIVED;
                             PARENT_DES.LAST_DERIVED := NEW_DES;
        when A_COMPONENT => null;
        when A_VARIABLE  => null;
      end case;
    end INSERT_SUBTYPE_INDICATOR_INFORMATION;

end SUBROUTINES_3_ROUTINES;
```

3.11.107 package ddl_names_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES, EXTRA_DEFINITIONS,
     SCHEMA_IO, KEYWORD_ROUTINES, SUBROUTINES_1_ROUTINES,
     SUBROUTINES_2_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES;
use IO_DEFINITIONS, DDL_DEFINITIONS, DDL_VARIABLES, EXTRA_DEFINITIONS,
     SCHEMA_IO, KEYWORD_ROUTINES, SUBROUTINES_1_ROUTINES,
     SUBROUTINES_2_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES;
```

```
package NAME_ROUTINES is

--      eof = end of file reached
--      eol = end of line ; reached
--      eoi = end of identifiers reached
--      comma = got a comma
--      valid_ident = got a valid identifier
--      invalid_ident = got an invalid identifier

  type IDENT_TYPE is (EOF, EOL, EOI, COMMA, VALID_IDENT, INVALID_IDENT);

  function VALID_QUALIFIED_IDENT_CHARS
          (THING   : in STRING;
           ERR_MSG : in BOOLEAN)
           return BOOLEAN;

  procedure VALID_NEW_TABLE_NAME
          (NAME       : in STRING;
           IDENT_DES  : in out ACCESS_IDENTIFIER_DESCRIPTOR;
           OK         : out BOOLEAN);

  function VALID_NEW_IDENT_NAME_DUPS_OK
          (NAME   : in STRING)
           return BOOLEAN;

  function VALID_NEW_IDENT_NAME
          (NAME   : in STRING)
           return BOOLEAN;

  function VALID_IDENT_CHARS
          (NAME   : in STRING)
           return BOOLEAN;

  function DUPLICATE_IDENT_NAME
          (NAME : in STRING)
           return BOOLEAN;

  function GOT_INVALID_CONSTRAINTS
          (NAME : in STRING)
           return BOOLEAN;

  procedure CHECK_EOF_EOL_IS_COMMA
           (NAME         : in STRING;
            RETURN_TYPE  : in out IDENT_TYPE);

  procedure CHECK_EOF_EOL_COLON_COMMA
           (NAME         : in STRING;
            RETURN_TYPE  : in out IDENT_TYPE);
```

```
  procedure VALID_NEW_TYPE_IDENT
           (NAME         : in STRING;
            RETURN_TYPE : in out IDENT_TYPE);

  procedure VALID_NEW_COMPONENT_IDENT
           (NAME         : in STRING;
            RETURN_TYPE : in out IDENT_TYPE);

  function VALID_NEW_PACKAGE_NAME
           (NAME : in STRING)
            return BOOLEAN;

  procedure VALID_NEW_SUBTYPE_IDENT
           (NAME         : in STRING;
            RETURN_TYPE : in out IDENT_TYPE);

  function VALID_NEW_FULL_COMPONENT_NAME
          (NAME        : in STRING;
           TABLE_NAME : in STRING)
           return BOOLEAN;

  function DUPLICATE_COMPONENT_NAME
          (NAME        : in STRING;
           TABLE_NAME : in STRING)
           return BOOLEAN;

  procedure VALID_NEW_VARIABLE_IDENT
           (NAME         : in STRING;
            RETURN_TYPE : in out IDENT_TYPE);

end NAME_ROUTINES;
```

3.11.108 package ddl_names.ada

```
package body NAME_ROUTINES is

---------------------------------------------------------------------
--
-- VALID_QUALIFIED_IDENT_CHARS
--
-- a valid qualified identifier may consist of only an identifier, or one or
-- two packages qualifying the identifier.  Errors are:
-- more than two package qualifiers
-- any character other than a-z 0-9 _ .
-- if a package or identifier begins with a character other than a-z

  function VALID_QUALIFIED_IDENT_CHARS
          (THING   : in STRING;
           ERR_MSG : in BOOLEAN)
           return BOOLEAN is
```

```
    FIRST : BOOLEAN := TRUE;
    CNT   : NATURAL := 0;
    C     : CHARACTER := ' ';
    VALID : BOOLEAN := TRUE;

  begin
    for I in THING'RANGE loop
      C := THING (I);
      if C not in 'A'..'Z' and then C not in '0'..'9' and then C /= '_'
           and then C /= '.' then
        VALID := FALSE;
      end if;
      if FIRST and C not in 'A'..'Z' then
        VALID := FALSE;
      end if;
      FIRST := FALSE;
      if C = '.' then
        FIRST := TRUE;
        CNT := CNT + 1;
        if CNT > 2 then
          VALID := FALSE;
        end if;
      end if;
    end loop;
    if VALID then
      return TRUE;
    else
      if ERR_MSG then
        PRINT_ERROR ("Invalid identifier: " & THING);
      end if;
      return FALSE;
    end if;
  end VALID_QUALIFIED_IDENT_CHARS;

---------------------------------------------------------------------
--
-- VALID_NEW_TABLE_NAME
--
-- given a new table identifier validate it, for characters and to see if it's
-- already been used or if it's a keyword.  It may have been used previously
-- as an identifier with different package names, in which case if the package
-- names are visible we should print a warning message.  If there is an
-- identifier descriptor for it return it.  If there is a matching table name
-- used by another schema with the same authorization id it's invalid.  It may
-- not contain the _not_null or _not_null_unique suffix, and may be no more than
-- 18 characters long.

  procedure VALID_NEW_TABLE_NAME
          (NAME       : in STRING;
```

```
          IDENT_DES : in out ACCESS_IDENTIFIER_DESCRIPTOR;
          OK        : out BOOLEAN) is

  TEST_FULL   : ACCESS_FULL_NAME_DESCRIPTOR := null;
  TEST_SCHEMA : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
  COUNT       : INTEGER := 0;
  IS_NULL     : BOOLEAN := FALSE;
  IS_UNIQUE   : BOOLEAN := FALSE;

begin
  OK := FALSE;
  IDENT_DES := null;
  if not VALID_IDENT_CHARS (NAME) then
    PRINT_ERROR ("Invalid table name" & NAME &
                 " - contains invalid characters");
    PRINT_TO_FILE ("    valid characters are A..Z, 0..9 and underscore");
    return;
  end if;
  if SQL_KEY_WORD (NAME) or ADA_KEY_WORD (NAME) then
    PRINT_ERROR ("Invalid table name " & NAME &
                 " - is SQL or ADA keyword");
    return;
  end if;
  if DUPLICATE_IDENT_NAME (NAME) then
    PRINT_ERROR ("Invalid table name " & NAME &
                 " - has already been used");
    return;
  end if;
  IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (NAME);
  IS_IDENTIFIER_NULL_OR_UNIQUE (NAME, IS_NULL, IS_UNIQUE);
  if IS_NULL then
    PRINT_ERROR ("Table names must not contain the " &
                "_NOT_NULL suffix");
    return;
  end if;
  if IS_UNIQUE then
    PRINT_ERROR ("Table names must not contain the " &
                "_NOT_NULL_UNIQUE suffix");
    return;
  end if;
  if NAME'LAST > 18 then
    PRINT_ERROR ("Table names must not be more than " &
                "18 characters long");
    return;
  end if;
  if IDENT_DES = null or CURRENT_SCHEMA_UNIT.AUTH_ID = null then
    OK := TRUE;
    return;
  end if;
```

```
    TEST_FULL := IDENT_DES.FIRST_FULL_NAME;
    while TEST_FULL /= null loop
      TEST_SCHEMA := TEST_FULL.SCHEMA_UNIT;
      if TEST_FULL.TYPE_IS.WHICH_TYPE = REC_ORD and then
         TEST_SCHEMA.AUTH_ID /= null and then
         TEST_SCHEMA.AUTH_ID.all = CURRENT_SCHEMA_UNIT.AUTH_ID.all then
        PRINT_ERROR ("Table name is also used in schema: " &
              STRING (TEST_SCHEMA.NAME.all));
        PRINT_TO_FILE ("         with the same authorization identifier");
        return;
      end if;
      TEST_FULL := TEST_FULL.NEXT_NAME;
    end loop;
    OK := TRUE;
  end VALID_NEW_TABLE_NAME;

------------------------------------------------------------------------
--
-- VALID_NEW_IDENT_NAME_DUPS_OK
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords

  function VALID_NEW_IDENT_NAME_DUPS_OK
          (NAME   : in STRING)
           return BOOLEAN is

  begin
    if not VALID_IDENT_CHARS (NAME) then
      PRINT_ERROR ("Invalid identifier: " & NAME &
                   " - contains invalid characters");
      PRINT_TO_FILE ("    valid characters are A..Z, 0..9 and underscore");
      return FALSE;
    end if;
    if (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST) /= STANDARD_NAME_ADA_SQL and
        OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST) /= DATABASE_NAME_ADA_SQL)
       and then (SQL_KEY_WORD (NAME) or ADA_KEY_WORD (NAME)) then
      PRINT_ERROR ("Invalid identifier: " & NAME &
                   " - is SQL or ADA keyword");
      return FALSE;
    end if;
    return TRUE;
  end VALID_NEW_IDENT_NAME_DUPS_OK;

------------------------------------------------------------------------
--
-- VALID_NEW_IDENT_NAME
```

```
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords
-- then check for a duplicate name

  function VALID_NEW_IDENT_NAME
          (NAME   : in STRING)
           return BOOLEAN is

  begin
    if not VALID_NEW_IDENT_NAME_DUPS_OK (NAME) then
      return FALSE;
    end if;
    if DUPLICATE_IDENT_NAME (NAME) then
      PRINT_ERROR ("Invalid identifier: " & NAME &
                   " - has already been used");
      return FALSE;
    end if;
    return TRUE;
  end VALID_NEW_IDENT_NAME;

--------------------------------------------------------------------------------
--
-- VALID_IDENT_CHARS
--
-- return false if first character is not A..Z and remaining characters aren't
-- A..Z 0..9 or _

  function VALID_IDENT_CHARS
          (NAME   : in STRING)
           return BOOLEAN is
  begin
    if NAME(NAME'FIRST) not in 'A'..'Z' then
      return FALSE;
    end if;
    for I in NAME'RANGE loop
      if NAME(I) in 'A'..'Z' or else
              NAME(I) in '0'..'9' or else
              NAME(I) = '_' then
        null;
      else
        return FALSE;
      end if;
    end loop;
    return TRUE;
  end VALID_IDENT_CHARS;
```

```
------------------------------------------------------------------
--
-- DUPLICATE_IDENT_NAME
--
-- if it's not in the identifier_descriptors it's looking good
-- if it is then we have to make sure that the package name in the full
-- name descriptor isn't duplicated.  if it was used previously
-- as an identifier with a different package name, then if the package
-- names are both visible print a warning message.

  function DUPLICATE_IDENT_NAME
          (NAME : in STRING)
           return BOOLEAN is

    IDENT_DES : ACCESS_IDENTIFIER_DESCRIPTOR := FIRST_IDENTIFIER;
    FULL_DES  : ACCESS_FULL_NAME_DESCRIPTOR := null;
    USED_DES  : ACCESS_USED_PACKAGE_DESCRIPTOR := null;
    COUNT     : INTEGER := 0;

  begin
    IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (NAME);
    if IDENT_DES /= null then
      FULL_DES := FIND_FULL_NAME_DESCRIPTOR
                 (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST), IDENT_DES);
      if FULL_DES /= null then
        return TRUE;
      else
        FULL_DES := IDENT_DES.FIRST_FULL_NAME;
        while FULL_DES /= null loop
          if FULL_DES.TABLE_NAME = null then
            USED_DES := CURRENT_SCHEMA_UNIT.FIRST_USED;
            while USED_DES /= null loop
              if FULL_DES.FULL_PACKAGE_NAME = USED_DES.NAME then
                if COUNT = 0 then
                  PRINT_TO_FILE ("Warning - identifier: " & NAME &
                                 " defined in both");
                  PRINT_TO_FILE ("          the current package: " &
                                 OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
                end if;
                PRINT_TO_FILE ("          and the visable package: " &
                               STRING (USED_DES.NAME.all));
                COUNT := COUNT + 1;
              end if;
              USED_DES := USED_DES.NEXT_USED;
            end loop;
          end if;
          FULL_DES := FULL_DES.NEXT_NAME;
        end loop;
      end if;
```

```
    end if;
    return FALSE;
  end DUPLICATE_IDENT_NAME;

-------------------------------------------------------------------
--
--  GOT_INVALID_CONSTRAINTS

  function GOT_INVALID_CONSTRAINTS
          (NAME : in STRING)
           return BOOLEAN is

    IS_NULL       : BOOLEAN := FALSE;
    IS_UNIQUE     : BOOLEAN := FALSE;

  begin
    IS_IDENTIFIER_NULL_OR_UNIQUE (NAME, IS_NULL, IS_UNIQUE);
    if IS_NULL then
      PRINT_ERROR ("Invalid identifier - _NOT_NULL suffix not permitted");
    elsif IS_UNIQUE then
      PRINT_ERROR ("Invalid identifier - _NOT_NULL_UNIQUE suffix " &
                   "not permitted");
    else
      return FALSE;
    end if;
    return TRUE;
  end GOT_INVALID_CONSTRAINTS;

----------------------------------------------------------------------
--
--  CHECK_EOF_EOL_IS_COMMA

  procedure CHECK_EOF_EOL_IS_COMMA
           (NAME        : in STRING;
            RETURN_TYPE : in out IDENT_TYPE) is
  begin
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS >= DONE then
      RETURN_TYPE := EOF;
    elsif GOT_END_OF_STATEMENT (NAME) then
      RETURN_TYPE := EOL;
    elsif NAME = "IS" then
      RETURN_TYPE := EOI;
    elsif NAME = "," then
      RETURN_TYPE := COMMA;
    else
      RETURN_TYPE := VALID_IDENT;
    end if;
  end CHECK_EOF_EOL_IS_COMMA;
```

```
-----------------------------------------------------------------------
--
--  CHECK_EOF_EOL_COLON_COMMA

  procedure CHECK_EOF_EOL_COLON_COMMA
           (NAME        : in STRING;
            RETURN_TYPE : in out IDENT_TYPE) is
  begin
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS >= DONE then
      RETURN_TYPE := EOF;
    elsif GOT_END_OF_STATEMENT (NAME) then
      RETURN_TYPE := EOL;
    elsif NAME = ":" then
      RETURN_TYPE := EOI;
    elsif NAME = "," then
      RETURN_TYPE := COMMA;
    else
      RETURN_TYPE := VALID_IDENT;
    end if;
  end CHECK_EOF_EOL_COLON_COMMA;

-----------------------------------------------------------------
--
-- VALID_NEW_TYPE_IDENT
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the IS return eoi
-- if it's a comma return comma
-- then check identifier for validity

  procedure VALID_NEW_TYPE_IDENT
           (NAME        : in STRING;
            RETURN_TYPE : in out IDENT_TYPE) is
  begin
    CHECK_EOF_EOL_IS_COMMA (NAME, RETURN_TYPE);
    if RETURN_TYPE /= VALID_IDENT then
      return;
    end if;
    if GOT_INVALID_CONSTRAINTS (NAME) then
      RETURN_TYPE := INVALID_IDENT;
      return;
    else
      null;
    end if;
    if not VALID_NEW_IDENT_NAME (NAME) then
      RETURN_TYPE := INVALID_IDENT;
    else
      null;
```

```
    end if;
  end VALID_NEW_TYPE_IDENT;

------------------------------------------------------------------
--
-- VALID_NEW_COMPONENT_IDENT
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the : return eoi
-- if it's a comma return comma
-- then check identifier for validity

  procedure VALID_NEW_COMPONENT_IDENT
           (NAME        : in STRING;
            RETURN_TYPE : in out IDENT_TYPE) is
  begin
    CHECK_EOF_EOL_COLON_COMMA (NAME, RETURN_TYPE);
    if RETURN_TYPE = VALID_IDENT and then
       (GOT_INVALID_CONSTRAINTS (NAME) or else
       not VALID_NEW_IDENT_NAME_DUPS_OK (NAME)) then
      RETURN_TYPE := INVALID_IDENT;
    else
      if NAME'LAST > 18 then
        PRINT_ERROR ("Invalid component identifier - max length is 18 " &
                     "characters");
        RETURN_TYPE := INVALID_IDENT;
      end if;
    end if;
  end VALID_NEW_COMPONENT_IDENT;

------------------------------------------------------------------
--
-- VALID_NEW_PACKAGE_NAME
--
-- If this is the first package declared
-- by the schema it may be anything but ADA_SQL.  If it is the second it
-- must be ADA_SQL.  If it is third or more we'll stuff it in the chain
-- no matter what it is but it's invalid.  Tell them it's invalid if it has
-- the suffix _NOT_NULL or _NOT_NULL_UNIQUE.

  function VALID_NEW_PACKAGE_NAME
           (NAME : in STRING)
            return BOOLEAN is

    NUMBER_OF_PACKAGES      : NATURAL := 0;
    NUMBER_OF_PACKAGES_OPEN : NATURAL := 0;

  begin
```

```
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE then
      PRINT_ERROR ("Invalid package declaration - premature eof");
    elsif not VALID_IDENT_CHARS (NAME) then
      PRINT_ERROR ("Invalid package name " & NAME &
                   " - contains invalid characters");
      PRINT_TO_FILE ("    valid characters are A..Z, 0..9 and underscore");
    elsif SQL_KEY_WORD (NAME) or ADA_KEY_WORD (NAME) then
      PRINT_ERROR ("Invalid package name " & NAME & " - is SQL or ADA keyword");
    elsif GOT_INVALID_CONSTRAINTS (NAME) then
      null;
    else
      GET_PACKAGE_COUNT (CURRENT_SCHEMA_UNIT, NUMBER_OF_PACKAGES,
                         NUMBER_OF_PACKAGES_OPEN);
      if NUMBER_OF_PACKAGES = 0 and then NAME = ADA_SQL_PACK then
        PRINT_ERROR ("Invalid package name -  outter package is ADA_SQL");
      elsif NUMBER_OF_PACKAGES = 1 and then NAME /= ADA_SQL_PACK then
        PRINT_ERROR ("Invalid package name - inner package isn't ADA_SQL");
      elsif NUMBER_OF_PACKAGES > 1 then
        PRINT_ERROR ("Invalid package declaration - schema unit may declare" &
                     " only two packages");
      end if;
      if CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE or
         CURRENT_SCHEMA_UNIT.AUTH_ID /= null or
         CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES or
         CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES or
         CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES or
         NUMBER_OF_PACKAGES /= NUMBER_OF_PACKAGES_OPEN then
        PRINT_ERROR ("Invalid package declaration - a package cannot " &
                     "be declared after an");
        PRINT_TO_FILE ("   authorization statement, after type or " &
                       "variable declarations, or after");
        PRINT_TO_FILE ("   an end package statement");
      end if;
      return TRUE;
    end if;
    return FALSE;
  end VALID_NEW_PACKAGE_NAME;

-----------------------------------------------------------------
--
-- VALID_NEW_SUBTYPE_IDENT
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the IS return eoi
-- if it's a comma return comma
-- then check identifier for validity

  procedure VALID_NEW_SUBTYPE_IDENT
```

```
          (NAME        : in STRING;
           RETURN_TYPE : in out IDENT_TYPE) is
begin
  CHECK_EOF_EOL_IS_COMMA (NAME, RETURN_TYPE);
  if RETURN_TYPE = VALID_IDENT and then
     not VALID_NEW_IDENT_NAME (NAME) then
    RETURN_TYPE := INVALID_IDENT;
  end if;
end VALID_NEW_SUBTYPE_IDENT;

------------------------------------------------------------------------
--
-- VALID_NEW_FULL_COMPONENT_NAME
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords
-- then check for a duplicate component name

  function VALID_NEW_FULL_COMPONENT_NAME
         (NAME       : in STRING;
          TABLE_NAME : in STRING)
          return BOOLEAN is

  begin
    if not VALID_NEW_IDENT_NAME_DUPS_OK (NAME) then
      return FALSE;
    end if;
    if DUPLICATE_COMPONENT_NAME (NAME, TABLE_NAME) then
      PRINT_ERROR ("Invalid identifier: " & NAME &
                   " - is already a component of table: " & TABLE_NAME);
      return FALSE;
    end if;
    return TRUE;
  end VALID_NEW_FULL_COMPONENT_NAME;

----------------------------------------------------------------
--
-- DUPLICATE_COMPONENT_NAME
--
-- if it's not in the identifier_descriptors it's looking good
-- if it is and the table names aren't the same than we're ok
-- if it is and the table names are the same, then we have to make sure
-- that the package name in the full name descriptor isn't duplicated.
-- if it was used previously as an identifier with a different package name,
-- but the same record name, then if the package names are both visible print
-- a warning message.
```

```
function DUPLICATE_COMPONENT_NAME
        (NAME       : in STRING;
         TABLE_NAME : in STRING)
         return BOOLEAN is

  IDENT_DES : ACCESS_IDENTIFIER_DESCRIPTOR := FIRST_IDENTIFIER;
  FULL_DES  : ACCESS_FULL_NAME_DESCRIPTOR := null;
  USED_DES  : ACCESS_USED_PACKAGE_DESCRIPTOR := null;
  COUNT     : INTEGER := 0;

begin
  IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (NAME);
  if IDENT_DES /= null then
    FULL_DES := FIND_FULL_NAME_COMPONENT_DESCRIPTOR
               (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST), IDENT_DES,
                TABLE_NAME);
    if FULL_DES /= null then
      return TRUE;
    else
      FULL_DES := IDENT_DES.FIRST_FULL_NAME;
      while FULL_DES /= null loop
        if FULL_DES.TABLE_NAME /= null and then
           STRING (FULL_DES.TABLE_NAME.all) = TABLE_NAME then
          USED_DES := CURRENT_SCHEMA_UNIT.FIRST_USED;
          while USED_DES /= null loop
            if FULL_DES.FULL_PACKAGE_NAME = USED_DES.NAME then
              if COUNT = 0 then
                PRINT_TO_FILE ("Warning - identifier: " & NAME &
                               " defined as component in table: " &
                               TABLE_NAME & " in both");
                PRINT_TO_FILE ("          the current package: " &
                               OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
              end if;
              PRINT_TO_FILE ("          and the visable package: " &
                             STRING (USED_DES.NAME.all));
              COUNT := COUNT + 1;
            end if;
            USED_DES := USED_DES.NEXT_USED;
          end loop;
        end if;
        FULL_DES := FULL_DES.NEXT_NAME;
      end loop;
    end if;
  end if;
  return FALSE;
end DUPLICATE_COMPONENT_NAME;

-----------------------------------------------------------------
--
```

```
-- VALID_NEW_VARIABLE_IDENT
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the : return eoi
-- if it's a comma return comma
-- then check identifier for validity
-- if it looks like an identifier but has constraints return invalid_identifier
-- if it really doesn't look like an identifier return unknown

  procedure VALID_NEW_VARIABLE_IDENT
           (NAME        : in STRING;
            RETURN_TYPE : in out IDENT_TYPE) is
  begin
    CHECK_EOF_EOL_COLON_COMMA (NAME, RETURN_TYPE);
    if RETURN_TYPE = VALID_IDENT and then
       (GOT_INVALID_CONSTRAINTS (NAME) or else
       not VALID_NEW_IDENT_NAME (NAME)) then
      RETURN_TYPE := INVALID_IDENT;
    end if;
  end VALID_NEW_VARIABLE_IDENT;

end NAME_ROUTINES;
```

3.11.109 package ddl_with_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES, SUBROUTINES_4_ROUTINES;
use  IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES, SUBROUTINES_4_ROUTINES;

package WITH_ROUTINES is

  procedure PROCESS_WITH;

end WITH_ROUTINES;
```

3.11.110 package ddl_with.ada

```
package body WITH_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_WITH
--
-- if the temp string is WITH and the WITHING flag is set, tell the user
--    that with is an invalid library unit name and don't process it
-- if the temp string is WITH and the WITHING flag is not set, then set it
```

```
--    if a package name had already been declared in the current schema or if
--       types or tables or variables have been declared tell them that
--       context clauses must be first, but go ahead and process the with
--       statement
--    return
-- if the temp string is a comma, just return
-- if the temp string is a semi colon change the WITHING flag to PROCESSING
--    and return
-- otherwise we have a library_unit_name to process

  procedure PROCESS_WITH is

    WITHED_UNIT_DES     : ACCESS_WITHED_UNIT_DESCRIPTOR := null;
    WITHED_UNIT_SCHEMA  : ACCESS_SCHEMA_UNIT_DESCRIPTOR := null;
    WITHED_HERE_BEFORE  : BOOLEAN := FALSE;
    PUT_ON_HOLD         : ACCESS_YET_TO_DO_DESCRIPTOR := null;
    NAME_STRING_LAST    : INTEGER := 1;
    NAME_STRING         : STRING (1..100) := (others => ' ');

  begin

-- process here if temp string = comma or semi colon or WITH

    if DEBUGGING then
      PRINT_TO_FILE ("*** WITH - schema unit: " &
                     STRING (CURRENT_SCHEMA_UNIT.NAME.ALL));
      PRINT_TO_FILE ("                 input: " &
                     TEMP_STRING (1..TEMP_STRING_LAST));
    end if;
    if TEMP_STRING (1..TEMP_STRING_LAST) = ";" then
      CURRENT_SCHEMA_UNIT.SCHEMA_STATUS := PROCESSING;
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) = "," then
      return;
    elsif TEMP_STRING (1..TEMP_STRING_LAST) = "WITH" then
      if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = WITHING then
        PRINT_ERROR ("Invalid library unit name: WITH - will " &
                     "not be processed");
      else
        CURRENT_SCHEMA_UNIT.SCHEMA_STATUS := WITHING;
        if CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /= null or else
           CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES or else
           CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES or else
           CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES then
          PRINT_ERROR ("Context clauses must appear before other " &
                       "declarations");
        end if;
      end if;
      return;
```

```
    end if;

-- do a withed library unit here:
-- get the withed library unit's schema if it's been declared before
-- find out if this schema unit has withed this library unit before
-- if we're trying to with ourselves tell the user and ignore this with

    NAME_STRING_LAST := TEMP_STRING_LAST;
    NAME_STRING (1..NAME_STRING_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
    case HOW_TO_DO_FILES is
      when UPPER_CASE => UPPER_CASE (NAME_STRING (1..NAME_STRING_LAST));
      when LOWER_CASE => LOWER_CASE (NAME_STRING (1..NAME_STRING_LAST));
      when AS_IS         => EXCHANGE_FOR_ORIGINAL (CURRENT_SCHEMA_UNIT,
                              NAME_STRING, NAME_STRING_LAST);
    end case;
    WITHED_UNIT_SCHEMA := FIND_SCHEMA_UNIT_DESCRIPTOR
                          (TEMP_STRING (1..TEMP_STRING_LAST));
    WITHED_HERE_BEFORE := DUPLICATE_WITH (CURRENT_SCHEMA_UNIT,
                                          WITHED_UNIT_SCHEMA);
    if WITHED_UNIT_SCHEMA = CURRENT_SCHEMA_UNIT then
      PRINT_ERROR ("Library Unit: " & TEMP_STRING(1..TEMP_STRING_LAST) &
        " - cannot with its self");
      return;
    end if;

-- if there is no schema for this with get a new schema, add it to the schema
--      chain, and set it's name
-- if it hasn't been withed before by the current schema unit then add it
--      to the chain of withed stuff
-- do not process the withed library unit name if it is schema_definition,
--      instead mark this one as done and continue with next
-- however if it is anything except schema-definition and this schema is an
--      authorization package tell the user that's not valid
-- if the status of the withing unit is already done then we don't have to do
--      anything else wth it

    if DEBUGGING then
      if WITHED_UNIT_SCHEMA = null then
        PRINT_TO_FILE ("    - new schema unit");
      else
        PRINT_TO_FILE ("    - old schema unit");
      end if;
      if WITHED_HERE_BEFORE then
        PRINT_TO_FILE ("    - withed here before");
      end if;
    end if;
    if WITHED_UNIT_SCHEMA = null then
      WITHED_UNIT_SCHEMA := GET_NEW_SCHEMA_UNIT_DESCRIPTOR;
      ADD_SCHEMA_UNIT_DESCRIPTOR (WITHED_UNIT_SCHEMA);
```

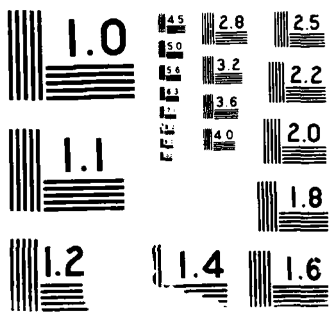
```
      WITHED_UNIT_SCHEMA.NAME := GET_NEW_LIBRARY_UNIT_NAME
                                    (TEMP_STRING (1..TEMP_STRING_LAST));
    end if;
    if not WITHED_HERE_BEFORE then
      WITHED_UNIT_DES := GET_NEW_WITHED_UNIT_DESCRIPTOR;
      WITHED_UNIT_DES.SCHEMA_UNIT := WITHED_UNIT_SCHEMA;
      ADD_WITHED_UNIT_DESCRIPTOR (WITHED_UNIT_DES, CURRENT_SCHEMA_UNIT);
    end if;
    if CHARACTER_STRINGS_MATCH (STRING (WITHED_UNIT_SCHEMA.NAME.all),
                                SCHEMA_DEF_NAME) then
      WITHED_UNIT_SCHEMA.SCHEMA_STATUS := DONE;
      if DEBUGGING then
        PRINT_TO_FILE ("    - schema definition");
      end if;
    elsif CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE then
      PRINT_ERROR ("The only library unit that may be withed by an " &
                   "authorization package");
      PRINT_TO_FILE ("     is " & SCHEMA_DEF_NAME);
    end if;
    if WITHED_UNIT_SCHEMA.SCHEMA_STATUS = DONE then
      return;
    end if;

-- put the current schema unit on hold (yet to do list)
-- set the withed unit schema as the current schema unit
-- then open the new current schema unit and return and process it

    PUT_ON_HOLD := GET_NEW_YET_TO_DO_DESCRIPTOR;
    PUT_ON_HOLD.UNDONE_SCHEMA := CURRENT_SCHEMA_UNIT;
    ADD_YET_TO_DO_DESCRIPTOR (PUT_ON_HOLD);
    CURRENT_SCHEMA_UNIT := WITHED_UNIT_SCHEMA;
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = NOTOPEN then
      CURRENT_SCHEMA_UNIT.NAME.all := LIBRARY_UNIT_NAME_STRING
            (NAME_STRING (1..NAME_STRING_LAST));
      OPEN_SCHEMA_UNIT (CURRENT_SCHEMA_UNIT);
      UPPER_CASE (STRING (CURRENT_SCHEMA_UNIT.NAME.all));
    end if;
    SET_UP_OUR_PACKAGE_NAME;
    SET_UP_WITH_USE_STANDARD_FOR_SCHEMA (CURRENT_SCHEMA_UNIT);
    return;
  end PROCESS_WITH;
end WITH_ROUTINES;
```

3.11.111 package ddl_auth_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_4_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
```

AD-A194 517 AN ADA/SQL (STRUCTURED QUERY LANGUAGE) APPLICATION SCANNER(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA B R BRYKCZYNSKI ET AL MAR 88 IAA-M-460 IDA/HQ-88-33317 6/6

UNCLASSIFIED MDA903-84-C-0031 F/G 12/5 NL


1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

```
    GET_NEW_DESCRIPTOR_ROUTINES, SEARCH_DESCRIPTOR_ROUTINES,
    SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES,
    SUBROUTINES_4_ROUTINES;

package SCHEMA_AUTHORIZATION_ROUTINES is

  procedure PROCESS_SCHEMA_AUTHORIZATION;

end SCHEMA_AUTHORIZATION_ROUTINES;
```

3.11.112 package ddl_auth.ada

```
package body SCHEMA_AUTHORIZATION_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_SCHEMA_AUTHORIZATION
--
-- on entry temp string is schema_authorization, it should be followed by
-- ": identifier :=" and the identifier.  It must be declared in an ADA_SQL
-- sub package and match the authorization identifier from an already
-- defined authorization package that was withed.  If types or tables have
-- already been declared warn the user that the schema authorizathion should
-- come first.  If variables have been declared tell them it's an error.

  procedure PROCESS_SCHEMA_AUTHORIZATION is

    AUTH_IDENTIFIER : STRING (1..250) := (others => ' ');
    AUTH_LAST       : NATURAL := 0;
    BUILD_STRING    : STRING (1..250) := (others => ' ');
    BUILD_LAST      : NATURAL := 0;
    DID_SCHEMA_DEF  : BOOLEAN := FALSE;
    DID_OTHERS      : BOOLEAN := FALSE;

  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** AUTH");
    end if;
    loop
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      exit when CURRENT_SCHEMA_UNIT.SCHEMA_STATUS >= DONE;
      BUILD_STRING (BUILD_LAST + 1..BUILD_LAST + TEMP_STRING_LAST) :=
                    TEMP_STRING (1..TEMP_STRING_LAST);
      BUILD_LAST := BUILD_LAST + TEMP_STRING_LAST;
      exit when BUILD_STRING (BUILD_LAST) = ';' ;
      exit when BUILD_LAST > 1 and then
              BUILD_STRING (BUILD_LAST-1 .. BUILD_LAST) = ":=" ;
    end loop;
    if BUILD_STRING (1..BUILD_LAST) /= ":IDENTIFIER:=" then
      PRINT_ERROR ("Invalid schema_authorization statement");
```

```
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
    return;
  end if;
  if CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE then
    PRINT_ERROR ("Cannot have schema_authorization declaration in an " &
                 "authorization package");
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
    return;
  end if;
  if CURRENT_SCHEMA_UNIT.AUTH_ID /= null then
    PRINT_ERROR ("Can define only one schema_authorization per schema unit");
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
    return;
  end if;
  GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
  AUTH_LAST := TEMP_STRING_LAST;
  AUTH_IDENTIFIER (1..AUTH_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
  if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
  end if;
  if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
    PRINT_ERROR ("Invalid schema_authorization statement");
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
    return;
  end if;
  if not SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE
                 (AUTH_IDENTIFIER (1..AUTH_LAST)) then
    PRINT_ERROR ("Schema_authorization identifier not found " &
                 "in a withed authorization package");
  end if;
  if not IN_ADA_SQL_PACKAGE then
    PRINT_ERROR ("Schema authorization statement must be in the " &
                 "ADA_SQL package");
  end if;
  if CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES or
     CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES then
    PRINT_ERROR ("Schema authorization statement must preceed " &
                 "type declarations");
  end if;
  if CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES then
    PRINT_ERROR ("Schema authorization statement not permitted in " &
                 "compilation unit defining");
    PRINT_TO_FILE ("   Ada/SQL program variables");
  end if;
  WITH_USE_SCHEMA_DEFINITION (DID_SCHEMA_DEF, DID_OTHERS);
  if not DID_SCHEMA_DEF then
    PRINT_ERROR ("Schema unit with authorization identifier must " &
                 "with and use Schema_definition");
  end if;
```

```
    CURRENT_SCHEMA_UNIT.AUTH_ID := GET_NEW_AUTH_IDENT_NAME
                                     (AUTH_IDENTIFIER (1..AUTH_LAST));
  end PROCESS_SCHEMA_AUTHORIZATION;

end SCHEMA_AUTHORIZATION_ROUTINES;
```

3.11.113 package ddl_function_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_4_ROUTINES, KEYWORD_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_4_ROUTINES, KEYWORD_ROUTINES;

package FUNCTION_ROUTINES is

  procedure PROCESS_FUNCTION;

end FUNCTION_ROUTINES;
```

3.11.114 package ddl_function.ada

```
package body FUNCTION_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_FUNCTION
--
-- on input temp string is function, it must be followed by an identifier
-- and then "is new authorization_identifier;" If it isn't it's invalid and
-- we don't accept an authorization identifier. If it is valid and an
-- authorization identifier has not already been declared in this schema unit
-- then this is it and set the flag that this is the auth package. If one has
-- already been declared in this schema unit then it's an error. If anything
-- in the with or use other than SCHEMA_DEFINITION that's an error.
-- One package must be open and none closed or it's an error. If we've
-- declared types or tables or variables it's an error. If it contains the
-- suffix _NOT_NULL or _NOT_NULL_UNIQUE it's an error and if it's more than
-- 18 characters long its an error

  procedure PROCESS_FUNCTION is

    AUTH_IDENTIFIER : STRING (1..250) := (others => ' ');
    AUTH_LAST       : NATURAL := 0;
    T_STRING        : STRING (1..250) := (others => ' ');
    T_LAST          : NATURAL := 0;
    BUILD_STRING    : STRING (1..250) := (others => ' ');
    BUILD_LAST      : NATURAL := 0;
    IS_NULL         : BOOLEAN := FALSE;
```

```
    IS_UNIQUE        : BOOLEAN := FALSE;
    DID_SCHEMA_DEF   : BOOLEAN := FALSE;
    DID_OTHERS       : BOOLEAN := FALSE;

  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** FUNCTION");
    end if;
    GET_STRING (CURRENT_SCHEMA_UNIT, AUTH_IDENTIFIER, AUTH_LAST);
    if DEBUGGING then
      PRINT_TO_FILE ("    - auth identifier: " &
                     AUTH_IDENTIFIER (1..AUTH_LAST));
    end if;
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE or else
       AUTH_IDENTIFIER (1..AUTH_LAST) = ";" then
      PRINT_ERROR ("Invalid function statement");
      return;
    end if;
    loop
      GET_STRING (CURRENT_SCHEMA_UNIT, T_STRING, T_LAST);
      if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE then
        PRINT_ERROR ("Invalid function statement");
        return;
      end if;
      BUILD_STRING (BUILD_LAST + 1..BUILD_LAST + T_LAST) :=
                    T_STRING(1..T_LAST);
      BUILD_LAST := BUILD_LAST + T_LAST;
      exit when T_STRING (1..T_LAST) = ";";
    end loop;
    if BUILD_STRING (1..BUILD_LAST) /= "ISNEWAUTHORIZATION_IDENTIFIER;" then
      PRINT_ERROR ("Invalid function statement");
      return;
    end if;
    if CURRENT_SCHEMA_UNIT.AUTH_ID = null then
      CURRENT_SCHEMA_UNIT.AUTH_ID := GET_NEW_AUTH_IDENT_NAME
                                     (AUTH_IDENTIFIER (1..AUTH_LAST));
      CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE := TRUE;
    else
      PRINT_ERROR ("Attempting to declare multiple " &
                   "authorization packages in a schema unit");
      return;
    end if;
    WITH_USE_SCHEMA_DEFINITION (DID_SCHEMA_DEF, DID_OTHERS);
    if not DID_SCHEMA_DEF or DID_OTHERS then
      PRINT_ERROR ("An authorization package withs and uses one " &
                   "library unit which");
      PRINT_TO_FILE ("   must be " & SCHEMA_DEF_NAME);
    end if;
    if CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /=
```

```
            CURRENT_SCHEMA_UNIT.LAST_DECLARED_PACKAGE or else
            CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE = null or else
            CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.FOUND_END then
      PRINT_ERROR ("An authorization package must declare only one " &
                   "package and the");
      PRINT_TO_FILE ("   authorization function must be in it");
    end if;
    if CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES or
       CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES or
       CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES then
      PRINT_ERROR ("An authorization package may declare only the " &
                   "authorization identifier");
    end if;
    IS_AUTH_ID_UNIQUE (AUTH_IDENTIFIER (1..AUTH_LAST), IS_UNIQUE);
    IS_IDENTIFIER_NULL_OR_UNIQUE (AUTH_IDENTIFIER (1..AUTH_LAST), IS_NULL,
             IS_UNIQUE);
    if IS_NULL then
      PRINT_ERROR ("An authorization identifier may not contain the " &
                   "_NOT_NULL suffix");
    end if;
    if IS_UNIQUE then
      PRINT_ERROR ("An authorization identifier may not contain the " &
                   "_NOT_NULL_UNIQUE suffix");
    end if;
    if AUTH_LAST > 18 then
      PRINT_ERROR ("An authorization identifier must not be more than " &
                   "18 characters in length");
    end if;
    if SQL_KEY_WORD (AUTH_IDENTIFIER (1..AUTH_LAST)) or
       ADA_KEY_WORD (AUTH_IDENTIFIER (1..AUTH_LAST)) then
      PRINT_ERROR ("An authorization identifier may not be a SQL or " &
                   "ADA keyword");
    end if;
  end PROCESS_FUNCTION;

end FUNCTION_ROUTINES;
```

3.11.115 package ddl_subroutines_2.ada

```
with NAME_ROUTINES;
use  NAME_ROUTINES;

package body SUBROUTINES_2_ROUTINES is

------------------------------------------------------------------
--
-- SPLIT_IDENT_2_PACKS
--
-- split up a string containing an identifier and possibly up to two
-- qualifying packages
```

```
procedure SPLIT_IDENT_2_PACKS
         (NAME              : in STRING;
          NAME_LAST         : in NATURAL;
          IDENT             : in out STRING;
          IDENT_LAST        : in out NATURAL;
          PACK1             : in out STRING;
          PACK1_LAST        : in out NATURAL;
          PACK2             : in out STRING;
          PACK2_LAST        : in out NATURAL;
          OK                : in out BOOLEAN;
          ERR_MSG           : in BOOLEAN) is

  CNT  : NATURAL := 0;
  DOT1 : NATURAL := 0;
  DOT2 : NATURAL := 0;

begin
  IDENT_LAST := 0;
  PACK1_LAST := 0;
  PACK2_LAST := 0;
  OK := VALID_QUALIFIED_IDENT_CHARS (NAME (1..NAME_LAST), ERR_MSG);
  if OK then
    for I in 1..NAME_LAST loop
      if NAME (I) = '.' then
        CNT := CNT + 1;
        if DOT1 = 0 then
          DOT1 := I;
        else
          DOT2 := I;
        end if;
      end if;
    end loop;
    if CNT = 0 then
      IDENT_LAST := NAME_LAST;
      IDENT (1..IDENT_LAST) := NAME (1..NAME_LAST);
    elsif CNT = 1 then
      PACK2_LAST := DOT1 - 1;
      PACK2 (1..PACK2_LAST) := NAME (1..DOT1 - 1);
      IDENT_LAST := NAME_LAST - DOT1;
      IDENT (1..IDENT_LAST) := NAME (DOT1 + 1..NAME_LAST);
    elsif CNT = 2 then
      PACK1_LAST := DOT1 - 1;
      PACK1 (1..PACK1_LAST) := NAME (1..DOT1 - 1);
      PACK2_LAST := DOT2 - DOT1 - 1;
      PACK2 (1..PACK2_LAST) := NAME (DOT1 + 1..DOT2 - 1);
      IDENT_LAST := NAME_LAST - DOT2;
      IDENT (1..IDENT_LAST) := NAME (DOT2 + 1..NAME_LAST);
    else
      OK := FALSE;
```

```
      end if;
      if (PACK1_LAST <= 0 and PACK2_LAST > 0) and then
        (PACK2 (1..PACK2_LAST) = STANDARD_NAME or
         PACK2 (1..PACK2_LAST) = CURSOR_NAME or
         PACK2 (1..PACK2_LAST) = DATABASE_NAME) then
           PACK1_LAST := PACK2_LAST;
           PACK1 (1..PACK1_LAST) := PACK2 (1..PACK2_LAST);
           PACK2_LAST := 0;
      end if;
    end if;
  end SPLIT_IDENT_2_PACKS;

----------------------------------------------------------------
--
-- FIND_IDENTIFIER_DESCRIPTOR
--
-- given an identifier return it's identifier_descriptor

  function FIND_IDENTIFIER_DESCRIPTOR
          (IDENTIFIER : in STRING)
           return ACCESS_IDENTIFIER_DESCRIPTOR is

    IDENT : ACCESS_IDENTIFIER_DESCRIPTOR := FIRST_IDENTIFIER;

  begin
    while IDENT /= null loop
      exit when STRING (IDENT.NAME.all) = IDENTIFIER;
      IDENT := IDENT.NEXT_IDENT;
    end loop;
    return IDENT;
  end FIND_IDENTIFIER_DESCRIPTOR;

----------------------------------------------------------------
--
-- FIND_FULL_NAME_COMPONENT_DESCRIPTOR
--
-- given an identifier's identifier_descriptor and a full package name
-- and a table name return the full_name_descriptor of a component or null
-- if it's not found

  function FIND_FULL_NAME_COMPONENT_DESCRIPTOR
          (PACK_NAME  : in STRING;
           IDENT      : in ACCESS_IDENTIFIER_DESCRIPTOR;
           TABLE_NAME : in STRING)
           return ACCESS_FULL_NAME_DESCRIPTOR is

    FULL : ACCESS_FULL_NAME_DESCRIPTOR := IDENT.FIRST_FULL_NAME;

  begin
```

```
    while FULL /= null loop
      if FULL.FULL_PACKAGE_NAME /= null and FULL.TABLE_NAME /= null then
        exit when STRING (FULL.FULL_PACKAGE_NAME.all) = PACK_NAME and
                  STRING (FULL.TABLE_NAME.all) = TABLE_NAME;
      end if;
      FULL := FULL.NEXT_NAME;
    end loop;
    return FULL;
  end FIND_FULL_NAME_COMPONENT_DESCRIPTOR;

----------------------------------------------------------------
--
-- FIND_FULL_NAME_DESCRIPTOR
--
-- given an identifier's identifier_descriptor and a full package name
-- return the full_name_descriptor or null if it's not found

  function FIND_FULL_NAME_DESCRIPTOR
          (PACK_NAME : in STRING;
           IDENT     : in ACCESS_IDENTIFIER_DESCRIPTOR)
           return ACCESS_FULL_NAME_DESCRIPTOR is

    FULL : ACCESS_FULL_NAME_DESCRIPTOR := IDENT.FIRST_FULL_NAME;

  begin
    while FULL /= null loop
      exit when STRING (FULL.FULL_PACKAGE_NAME.all) = PACK_NAME and
                FULL.TABLE_NAME = null;
      FULL := FULL.NEXT_NAME;
    end loop;
    return FULL;
  end FIND_FULL_NAME_DESCRIPTOR;

-----------------------------------------------------------------
--
-- GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR
--
-- given the identifier descriptor and possible known outter and inner
-- packages and possible trying outter and inner packages set up to create
-- the full package name to look for in the full name descriptors.
-- there must be at least one outter and one inner package.  the known ones
-- must be used if available and if there are corresponding try ones they
-- better match.

  function GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR
          (IDENT_DES            : in ACCESS_IDENTIFIER_DESCRIPTOR;
           TRY_OUTTER           : in STRING;
           TRY_OUTTER_LAST      : in NATURAL;
           TRY_INNER            : in STRING;
```

```
          TRY_INNER_LAST      : in NATURAL;
          KNOWN_OUTTER        : in STRING;
          KNOWN_OUTTER_LAST   : in NATURAL;
          KNOWN_INNER         : in STRING;
          KNOWN_INNER_LAST    : in NATURAL)
          return ACCESS_FULL_NAME_DESCRIPTOR is

  FULL_NULL              : ACCESS_FULL_NAME_DESCRIPTOR := null;
  DO_OUTTER              : STRING (1..250) := (others => ' ');
  DO_OUTTER_LAST         : NATURAL := 0;
  DO_INNER               : STRING (1..250) := (others => ' ');
  DO_INNER_LAST          : NATURAL := 0;
  SPECIAL                : BOOLEAN := FALSE;

begin
  if KNOWN_INNER_LAST = 0 and then TRY_INNER_LAST = 0 and then
     ((KNOWN_OUTTER (1..KNOWN_OUTTER_LAST) = STANDARD_NAME or
       TRY_OUTTER (1..TRY_OUTTER_LAST) = STANDARD_NAME) or
      (KNOWN_OUTTER (1..KNOWN_OUTTER_LAST) = CURSOR_NAME or
       TRY_OUTTER (1..TRY_OUTTER_LAST) = CURSOR_NAME) or
      (KNOWN_OUTTER (1..KNOWN_OUTTER_LAST) = DATABASE_NAME or
       TRY_OUTTER (1..TRY_OUTTER_LAST) = DATABASE_NAME)) then
    SPECIAL := TRUE;
  end if;
  if ((KNOWN_OUTTER_LAST < 1 and TRY_OUTTER_LAST < 1) or
     (KNOWN_INNER_LAST < 1 and TRY_INNER_LAST < 1 and not SPECIAL) or
     ((KNOWN_OUTTER_LAST > 0 and TRY_OUTTER_LAST > 0) and then
           (TRY_OUTTER (1..TRY_OUTTER_LAST) /=
           KNOWN_OUTTER (1..KNOWN_OUTTER_LAST))) or
     ((KNOWN_INNER_LAST > 0 and TRY_INNER_LAST > 0) and then
           (TRY_INNER (1..TRY_INNER_LAST) /=
           KNOWN_INNER (1..KNOWN_INNER_LAST)))) then
    return FULL_NULL;
  end if;
  if KNOWN_OUTTER_LAST > 0 then
    DO_OUTTER_LAST := KNOWN_OUTTER_LAST;
    DO_OUTTER (1..DO_OUTTER_LAST) := KNOWN_OUTTER (1..KNOWN_OUTTER_LAST);
  elsif TRY_OUTTER_LAST > 0 then
    DO_OUTTER_LAST := TRY_OUTTER_LAST;
    DO_OUTTER (1..DO_OUTTER_LAST) := TRY_OUTTER (1..TRY_OUTTER_LAST);
  end if;
  if KNOWN_INNER_LAST > 0 then
    DO_INNER_LAST := KNOWN_INNER_LAST + 1;
    DO_INNER (1) := '.';
    DO_INNER (2..DO_INNER_LAST) := KNOWN_INNER (1..KNOWN_INNER_LAST);
  elsif TRY_INNER_LAST > 0 then
    DO_INNER_LAST := TRY_INNER_LAST + 1;
    DO_INNER (1) := '.';
    DO_INNER (2..DO_INNER_LAST) := TRY_INNER (1..TRY_INNER_LAST);
```

```
    end if;
    return FIND_FULL_NAME_DESCRIPTOR ((DO_OUTTER (1..DO_OUTTER_LAST)
                & DO_INNER (1..DO_INNER_LAST)), IDENT_DES);
  end GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR;

-----------------------------------------------------------------
--
-- FIND_FULL_NAME_DESCRIPTOR_VISIBLE
--
-- given current schema, identifier's descriptor and either no package names,
-- both the inner and outter package name or only the inner package name
-- of only the outter if its one of the special (database, standard,
-- cursor_definition) find the full name descriptor that would be
-- visible from current schema.  First choice is current package.  If no match
-- then next choice is from packages currently used (it's already been
-- established at this point that we're two levels deep into packages unless
-- we're doing one of the special ones).  If it isn't found yet then we have
-- to search the withed list, but in that case the full package name better
-- be described.

  function FIND_FULL_NAME_DESCRIPTOR_VISIBLE
          (SCHEMA          : in ACCESS_SCHEMA_UNIT_DESCRIPTOR;
           IDENT_DES       : in ACCESS_IDENTIFIER_DESCRIPTOR;
           OUTTER_PACKAGE  : in STRING;
           OUTTER_LAST     : in NATURAL;
           INNER_PACKAGE   : in STRING;
           INNER_LAST      : in NATURAL)
           return ACCESS_FULL_NAME_DESCRIPTOR is

    FULL                    : ACCESS_FULL_NAME_DESCRIPTOR :=
                              IDENT_DES.FIRST_FULL_NAME;
    FULL_HOLD               : ACCESS_FULL_NAME_DESCRIPTOR := null;
    USED                    : ACCESS_USED_PACKAGE_DESCRIPTOR := null;
    WITHED                  : ACCESS_WITHED_UNIT_DESCRIPTOR := null;
    TRY_OUTTER              : STRING (1..250) := (others => ' ');
    TRY_OUTTER_LAST         : NATURAL := 0;
    TRY_INNER               : STRING (1..250) := (others => ' ');
    TRY_INNER_LAST          : NATURAL := 0;

  begin
    SPLIT_PACKAGE_NAME (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST),
                        TRY_OUTTER, TRY_OUTTER_LAST, TRY_INNER,
                        TRY_INNER_LAST);
    FULL := GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR (IDENT_DES, TRY_OUTTER,
              TRY_OUTTER_LAST, TRY_INNER, TRY_INNER_LAST, OUTTER_PACKAGE,
              OUTTER_LAST, INNER_PACKAGE, INNER_LAST);
    if FULL /= null then
      return FULL;
    end if;
```

```
    USED := SCHEMA.FIRST_USED;
    while USED /= null loop
      SPLIT_PACKAGE_NAME (STRING(USED.NAME.all),TRY_OUTTER,
                          TRY_OUTTER_LAST, TRY_INNER, TRY_INNER_LAST);
      FULL := GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR (IDENT_DES, TRY_OUTTER,
              TRY_OUTTER_LAST, TRY_INNER, TRY_INNER_LAST, OUTTER_PACKAGE,
              OUTTER_LAST, INNER_PACKAGE, INNER_LAST);
      if FULL /= null then
        if FULL_HOLD = null then
          FULL_HOLD := FULL;
        else
          FULL := null;
          return FULL;
        end if;
      end if;
      USED := USED.NEXT_USED;
    end loop;
    if FULL_HOLD /= null then
      return FULL_HOLD;
    end if;
    WITHED := SCHEMA.FIRST_WITHED;
    while WITHED /= null loop
      SPLIT_PACKAGE_NAME (STRING(WITHED.SCHEMA_UNIT.NAME.all),TRY_OUTTER,
                          TRY_OUTTER_LAST, TRY_INNER, TRY_INNER_LAST);
      if OUTTER_PACKAGE (1..OUTTER_LAST) = TRY_OUTTER (1..TRY_OUTTER_LAST)
         and INNER_PACKAGE (1..INNER_LAST) = TRY_INNER (1..TRY_INNER_LAST)
         then
        FULL := GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR (IDENT_DES,
             TRY_OUTTER, TRY_OUTTER_LAST, TRY_INNER, TRY_INNER_LAST,
             OUTTER_PACKAGE, OUTTER_LAST, INNER_PACKAGE, INNER_LAST);
        if FULL /= null then
          return FULL;
        end if;
      end if;
      WITHED := WITHED.NEXT_WITHED;
    end loop;
    FULL := null;
    return FULL;
  end FIND_FULL_NAME_DESCRIPTOR_VISIBLE;

------------------------------------------------------------------
--
-- BASE_TYPE_INTEGER

  procedure BASE_TYPE_INTEGER
           (FULL_DES      : in ACCESS_FULL_NAME_DESCRIPTOR;
            IS_INT        : out BOOLEAN;
            LO_RANGE      : out INT;
            HI_RANGE      : out INT) is
```

```
  begin
    LO_RANGE := -1;
    HI_RANGE := -1;
    IS_INT   := FALSE;
    if FULL_DES.TYPE_IS.WHICH_TYPE = INT_EGER then
      LO_RANGE := FULL_DES.TYPE_IS.RANGE_LO_INT;
      HI_RANGE := FULL_DES.TYPE_IS.RANGE_HI_INT;
      IS_INT := TRUE;
    end if;
  end BASE_TYPE_INTEGER;

------------------------------------------------------------------------
--
-- LOCATE_PREVIOUS_IDENTIFIER
--
-- given an identifier, possibly qualified return it's identifier descriptor
-- and it's full name descriptor.  Error = 0 = ok
-- error 1 = it is not a valid qualified identifier
-- error 2 = does not split correctly into 2 packages and 1 identifier
--           maybe invalid nesting of packages
-- error 3 = cannot find identifier by this name
-- error 4 = cannot identify unique full name identifier of this name

  procedure LOCATE_PREVIOUS_IDENTIFIER
          (FULL_IDENT      : in out STRING;
           FULL_IDENT_LAST : in out NATURAL;
           IDENT_DES       : in out ACCESS_IDENTIFIER_DESCRIPTOR;
           FULL_DES        : in out ACCESS_FULL_NAME_DESCRIPTOR;
           ERROR           : in out INTEGER;
           ERR_MSG         : in BOOLEAN) is

    OK         : BOOLEAN := TRUE;
    IDENT      : STRING (1..250) := (others => ' ');
    PACK1      : STRING (1..250) := (others => ' ');
    PACK2      : STRING (1..250) := (others => ' ');
    IDENT_LAST : NATURAL := 0;
    PACK1_LAST : NATURAL := 0;
    PACK2_LAST : NATURAL := 0;

  begin
    ERROR := 0;
    if not VALID_QUALIFIED_IDENT_CHARS (FULL_IDENT (1..FULL_IDENT_LAST),
           ERR_MSG) then
      ERROR := 1;
    else
      SPLIT_IDENT_2_PACKS (FULL_IDENT, FULL_IDENT_LAST, IDENT, IDENT_LAST,
                           PACK1, PACK1_LAST, PACK2, PACK2_LAST, OK, ERR_MSG);
      if not OK then
        ERROR := 2;
```

```
        else
          IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (IDENT (1..IDENT_LAST));
          if IDENT_DES = null then
            ERROR := 3;
          else
            FULL_DES := FIND_FULL_NAME_DESCRIPTOR_VISIBLE (CURRENT_SCHEMA_UNIT,
                        IDENT_DES, PACK1 (1..PACK1_LAST), PACK1_LAST,
                        PACK2 (1..PACK2_LAST), PACK2_LAST);
            if FULL_DES = null then
              ERROR := 4;
            end if;
          end if;
        end if;
      end if;
    end LOCATE_PREVIOUS_IDENTIFIER;
  ----------------------------------------------------------------------
  --
  -- STRING_TO_INT

    procedure STRING_TO_INT
            (INT_STRING : in STRING;
             OK         : out BOOLEAN;
             OUT_INT    : out INT) is

      TEMP : INT := 0;

    begin
      OUT_INT := 0;
      OUT_INT := INT'VALUE (INT_STRING);
      TEMP := INT'VALUE (INT_STRING);
      OK := TRUE;
    exception
      when CONSTRAINT_ERROR => OK := FALSE;
        OUT_INT := 0;
    end STRING_TO_INT;

  ----------------------------------------------------------------------
  --
  -- BASE_TYPE_CHAR
  --
  -- given a full_name descriptor find out if it's base type is character

    function BASE_TYPE_CHAR
            (FULL_DES : in ACCESS_FULL_NAME_DESCRIPTOR)
             return BOOLEAN is

      VALID : BOOLEAN := FALSE;

    begin
```

```
    VALID := (FULL_DES.TYPE_IS.WHICH_TYPE = ENUMERATION and then
            STRING (FULL_DES.NAME.all) = CHARACTER_BASE and then
            STRING (FULL_DES.FULL_PACKAGE_NAME.all) = STANDARD_NAME_ADA_SQL) or
           (FULL_DES.TYPE_IS.WHICH_TYPE = ENUMERATION and then
            STRING (FULL_DES.TYPE_IS.ULT_PARENT_TYPE.FULL_NAME.NAME.all) =
                             CHARACTER_BASE and then STRING
           (FULL_DES.TYPE_IS.ULT_PARENT_TYPE.FULL_NAME.FULL_PACKAGE_NAME.all) =
                             STANDARD_NAME_ADA_SQL);
    return VALID;
  end BASE_TYPE_CHAR;

--------------------------------------------------------------------
--
-- IS_IDENTIFIER_NULL_OR_UNIQUE
--

  procedure IS_IDENTIFIER_NULL_OR_UNIQUE
          (THING     : in STRING;
           IS_NULL   : out BOOLEAN;
           IS_UNIQUE : out BOOLEAN) is

    LAST : INTEGER := 0;

  begin
    IS_NULL := FALSE;
    IS_UNIQUE := FALSE;
    LAST := THING'LAST;
    if LAST > SUF_NOT_NULL_LEN and then
       THING (LAST - (SUF_NOT_NULL_LEN - 1)..LAST) = SUF_NOT_NULL then
      IS_NULL := TRUE;
    elsif LAST > SUF_UNIQUE_LEN and then
      THING (LAST - (SUF_UNIQUE_LEN - 1)..LAST) = SUF_UNIQUE then
      IS_UNIQUE := TRUE;
    end if;
  end IS_IDENTIFIER_NULL_OR_UNIQUE;

--------------------------------------------------------------
--
-- IN_ADA_SQL_PACKAGE

  function IN_ADA_SQL_PACKAGE
           return BOOLEAN is

    OUTTER      : STRING (1..250) := (others => ' ');
    OUTTER_LAST : NATURAL := 0;
    INNER       : STRING (1..250) := (others => ' ');
    INNER_LAST  : NATURAL := 0;

  begin
```

```
    if OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST) = STANDARD_NAME_ADA_SQL or
       OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST) = DATABASE_NAME_ADA_SQL or
       OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST) = CURSOR_NAME_ADA_SQL then
        return TRUE;
    end if;
    if OUR_PACKAGE_NAME_LAST > ADA_SQL_PACK'LAST + 1 then
      SPLIT_PACKAGE_NAME (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST),
            OUTTER, OUTTER_LAST, INNER, INNER_LAST);
    end if;
    return ((OUTTER_LAST > 0 and
           INNER_LAST > 0) and then
           INNER (1..INNER_LAST) = ADA_SQL_PACK);
  end IN_ADA_SQL_PACKAGE;

------------------------------------------------------------------------
--
-- ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
--
-- ident descriptor may already exist, if not create on
-- full des will not already exist, create it

  procedure ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES        : in out ACCESS_IDENTIFIER_DESCRIPTOR;
            FULL_DES         : in out ACCESS_FULL_NAME_DESCRIPTOR;
            NAME             : in STRING) is
  begin
    if IDENT_DES = null then
      IDENT_DES := GET_NEW_IDENTIFIER_DESCRIPTOR;
      IDENT_DES.NAME := GET_NEW_TYPE_NAME (NAME);
      ADD_IDENTIFIER_DESCRIPTOR (IDENT_DES);
    end if;
    FULL_DES := GET_NEW_FULL_NAME_DESCRIPTOR;
    FULL_DES.NAME := GET_NEW_TYPE_NAME (NAME);
    FULL_DES.FULL_PACKAGE_NAME := GET_NEW_PACKAGE_NAME
                         (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
    IS_IDENTIFIER_NULL_OR_UNIQUE (NAME, FULL_DES.IS_NOT_NULL,
                                  FULL_DES.IS_NOT_NULL_UNIQUE);
    FULL_DES.SCHEMA_UNIT := CURRENT_SCHEMA_UNIT;
    ADD_FULL_NAME_DESCRIPTOR (FULL_DES, IDENT_DES);
  end ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS;

------------------------------------------------------------------------
--
-- ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS
--
-- ident descriptor may already exist, if not create on
-- full des will not already exist, create it

  procedure ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS
```

```
            (IDENT_DES          : in out ACCESS_IDENTIFIER_DESCRIPTOR;
             FULL_DES           : in out ACCESS_FULL_NAME_DESCRIPTOR;
             NAME               : in STRING;
             TABLE_NAME         : in STRING) is
  begin
    if IDENT_DES = null then
      IDENT_DES := GET_NEW_IDENTIFIER_DESCRIPTOR;
      IDENT_DES.NAME := GET_NEW_TYPE_NAME (NAME);
      ADD_IDENTIFIER_DESCRIPTOR (IDENT_DES);
    end if;
    FULL_DES := GET_NEW_FULL_NAME_DESCRIPTOR;
    FULL_DES.NAME := GET_NEW_TYPE_NAME (NAME);
    FULL_DES.FULL_PACKAGE_NAME := GET_NEW_PACKAGE_NAME
                         (OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
    FULL_DES.TABLE_NAME := GET_NEW_RECORD_NAME (TABLE_NAME);
    IS_IDENTIFIER_NULL_OR_UNIQUE (NAME, FULL_DES.IS_NOT_NULL,
                                  FULL_DES.IS_NOT_NULL_UNIQUE);
    FULL_DES.SCHEMA_UNIT := CURRENT_SCHEMA_UNIT;
    ADD_FULL_NAME_DESCRIPTOR (FULL_DES, IDENT_DES);
  end ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS;

end SUBROUTINES_2_ROUTINES;
```

3.11.116 package ddl_package_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, KEYWORD_ROUTINES, SUBROUTINES_2_ROUTINES,
     NAME_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, KEYWORD_ROUTINES, SUBROUTINES_2_ROUTINES,
     NAME_ROUTINES;

package PACKAGE_ROUTINES is

  procedure PROCESS_PACKAGE;

end PACKAGE_ROUTINES;
```

3.11.117 package ddl_package.ada

```
package body PACKAGE_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_PACKAGE
--
-- the token we get in temp string is "package" toss it, then read the
-- identifier and set the pointers.  If this is the first package declared
```

```
-- by the schema it may be anything but ADA_SQL.  If it is the second it
-- must be ADA_SQL.  If it is third or more we'll stuff it in the chain
-- no matter what it is but it's invalid.  Tell them it's invalid if it has
-- the suffix _NOT_NULL or _NOT_NULL_UNIQUE. Gobble up the "is" after the
-- identifier too

  procedure PROCESS_PACKAGE is

    PACKAGE_NAME       : STRING (1..250) := (others => ' ');
    PACKAGE_NAME_LAST  : NATURAL := 0;
    PACK_DES           : ACCESS_DECLARED_PACKAGE_DESCRIPTOR := null;
    NUMBER_OF_PACKAGES : NATURAL := 0;
    IS_NULL            : BOOLEAN := FALSE;
    IS_UNIQUE          : BOOLEAN := FALSE;

  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** PACKAGE");
    end if;
    GET_STRING (CURRENT_SCHEMA_UNIT, PACKAGE_NAME, PACKAGE_NAME_LAST);
    if VALID_NEW_PACKAGE_NAME (PACKAGE_NAME (1..PACKAGE_NAME_LAST)) then
      PACK_DES := GET_NEW_DECLARED_PACKAGE_DESCRIPTOR;
      PACK_DES.NAME := GET_NEW_PACKAGE_NAME (PACKAGE_NAME (1..PACKAGE_NAME_LAST))
      ADD_DECLARED_PACKAGE_DESCRIPTOR (PACK_DES, CURRENT_SCHEMA_UNIT);
      SET_UP_OUR_PACKAGE_NAME;
      if DEBUGGING then
        PRINT_TO_FILE ("    - our package name now: " &
                       OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
      end if;
      GET_STRING (CURRENT_SCHEMA_UNIT, PACKAGE_NAME, PACKAGE_NAME_LAST);
      if PACKAGE_NAME(1..PACKAGE_NAME_LAST) /= "IS" then
        PRINT_ERROR ("Incomplete package declaration - package name must be " &
                     "followed by IS");
      end if;
    end if;
  end PROCESS_PACKAGE;

end PACKAGE_ROUTINES;
```

3.11.118 package ddl_list_spec.ada

```
with DDL_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES, EXTRA_DEFINITIONS,
     SUBROUTINES_2_ROUTINES, NAME_ROUTINES;
use  DDL_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES, EXTRA_DEFINITIONS,
     SUBROUTINES_2_ROUTINES, NAME_ROUTINES;

package LIST_ROUTINES is

  function MAKE_LIST_OF_NAMES
          return BOOLEAN;
```

```
  procedure ADD_NAME_TO_PROCESS_LIST
          (NEW_NAME_TO_PROCESS_LIST : in out ACCESS_NAME_TO_PROCESS_LIST);

  function GET_NEW_LIST_NAME
         (TEMP : in STRING)
          return LIST_NAME;

  function GET_NEW_NAME_TO_PROCESS_LIST
          return ACCESS_NAME_TO_PROCESS_LIST;

  function MAKE_LIST_OF_COMPONENTS
          return BOOLEAN;

  procedure ADD_COMPONENT_TO_PROCESS_LIST
          (NEW_COMPONENT_TO_PROCESS_LIST : in out
                        ACCESS_COMPONENT_TO_PROCESS_LIST);

  function GET_NEW_LIST_COMPONENT
         (TEMP : in STRING)
          return LIST_COMPONENT;

  function GET_NEW_COMPONENT_TO_PROCESS_LIST
          return ACCESS_COMPONENT_TO_PROCESS_LIST;

  function MAKE_LIST_OF_VARIABLES
          return BOOLEAN;

end LIST_ROUTINES;
```

3.11.119 package ddl_list.ada

```
package body LIST_ROUTINES is

-----------------------------------------------------------------
--
-- MAKE_LIST_OF_NAMES
--
-- the next read should point us to a name of a type, derived type or subtype
-- we want to chain up a list of them to process later
-- stop when we find IS or ;
-- temp string will contain TYPE or SUBTYPE on entry
-- identifier is invalid if TYPE declaration and suffix of _NOT_NULL or
-- _NOT_NULL_UNIQUE

  function MAKE_LIST_OF_NAMES
          return BOOLEAN is

    NEW_NAME     : ACCESS_NAME_TO_PROCESS_LIST := null;
    NEED_COMMA   : BOOLEAN := FALSE;
    TYPE_SUBTYPE : STRING (1..7) := "subtype";
```

```
    T_S_LEN       : NATURAL := 7;
    WHICH_IDENT   : IDENT_TYPE := INVALID_IDENT;
    IS_TYPE       : BOOLEAN := FALSE;

  begin
    FIRST_NAME_TO_PROCESS := null;
    LAST_NAME_TO_PROCESS  := null;
    NEED_COMMA            := FALSE;
    if TEMP_STRING(1..TEMP_STRING_LAST) = "TYPE" then
      T_S_LEN := 4;
      TYPE_SUBTYPE (1..T_S_LEN) := "type";
      IS_TYPE := TRUE;
    end if;
    loop
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if IS_TYPE then
        VALID_NEW_TYPE_IDENT (TEMP_STRING (1..TEMP_STRING_LAST), WHICH_IDENT);
      else
        VALID_NEW_SUBTYPE_IDENT (TEMP_STRING (1..TEMP_STRING_LAST),
                                 WHICH_IDENT);
      end if;
      case WHICH_IDENT is
        when EOF => PRINT_ERROR ("Incomplete " & TYPE_SUBTYPE (1..T_S_LEN) &
                                 " declaration - premature eof found");
                    return FALSE;
        when EOL => PRINT_ERROR ("Incomplete " & TYPE_SUBTYPE (1..T_S_LEN) &
                                 " declaration - premature ; found");
                    return FALSE;
        when EOI => if FIRST_NAME_TO_PROCESS = null then
                      PRINT_ERROR ("Incomplete " & TYPE_SUBTYPE (1..T_S_LEN) &
                                   " declaration - no valid identifiers");
                      return FALSE;
                    else
                      return TRUE;
                    end if;
        when COMMA => if not NEED_COMMA then
                        PRINT_ERROR ("Invalid " & TYPE_SUBTYPE (1..T_S_LEN) &
                                     " declaration - extra comma");
                      end if;
                      NEED_COMMA := FALSE;
        when INVALID_IDENT => null;
        when VALID_IDENT => if NEED_COMMA then
                              PRINT_ERROR ("Invalid " &
                                           TYPE_SUBTYPE (1..T_S_LEN)
                                           & " declaration - missing comma");
                            end if;
                            NEED_COMMA := TRUE;
                            NEW_NAME := GET_NEW_NAME_TO_PROCESS_LIST;
                            NEW_NAME.NAME := GET_NEW_LIST_NAME
```

```
                                  (TEMP_STRING (1..TEMP_STRING_LAST));
                        ADD_NAME_TO_PROCESS_LIST (NEW_NAME);
    end case;
  end loop;
end MAKE_LIST_OF_NAMES;

-----------------------------------------------------------------------------
--
-- ADD_NAME_TO_PROCESS_LIST
--
-- if this is the first name-to-process set the first pointer
-- otherwise set the "next" pointer in the previously last name-to-process to
--          point to this new name-to-process
-- set the previous pointer in this new name-to-process to point to the
--          old last name-to-process
-- and now the new name-to-process is the last one

  procedure ADD_NAME_TO_PROCESS_LIST
          (NEW_NAME_TO_PROCESS_LIST : in out ACCESS_NAME_TO_PROCESS_LIST) is
  begin
    if LAST_NAME_TO_PROCESS = null then
      FIRST_NAME_TO_PROCESS := NEW_NAME_TO_PROCESS_LIST;
    else
      LAST_NAME_TO_PROCESS.NEXT_NAME := NEW_NAME_TO_PROCESS_LIST;
    end if;
    NEW_NAME_TO_PROCESS_LIST.PREVIOUS_NAME := LAST_NAME_TO_PROCESS;
    LAST_NAME_TO_PROCESS := NEW_NAME_TO_PROCESS_LIST;
  end ADD_NAME_TO_PROCESS_LIST;

-----------------------------------------------------------------------------
--
-- GET_NEW_LIST_NAME
--

  function GET_NEW_LIST_NAME
          (TEMP : in STRING)
           return LIST_NAME is
  begin
    return new LIST_NAME_STRING' (LIST_NAME_STRING (TEMP));
  end GET_NEW_LIST_NAME;

-----------------------------------------------------------------------------
--
-- GET_NEW_NAME_TO_PROCESS_LIST
--

  function GET_NEW_NAME_TO_PROCESS_LIST
           return ACCESS_NAME_TO_PROCESS_LIST is
  begin
```

```
    return new NAME_TO_PROCESS_LIST'
       (NAME              => null,
        PREVIOUS_NAME     => null,
        NEXT_NAME         => null);
  end GET_NEW_NAME_TO_PROCESS_LIST;

-----------------------------------------------------------------
--
-- MAKE_LIST_OF_COMPONENTS
--
-- on entry we should point to a component of a record type
-- we want to chain up a list of them to process later
-- stop when we find : or ;
-- temp string will contain a component name on entry
-- they must not contain _NOT_NULL or _NOT_NULL_UNIQUE suffixes and must be no
-- more than 18 characters long

  function MAKE_LIST_OF_COMPONENTS
           return BOOLEAN is

    NEW_COMPONENT      : ACCESS_COMPONENT_TO_PROCESS_LIST := null;
    NEED_COMMA         : BOOLEAN := FALSE;
    WHICH_IDENT        : IDENT_TYPE := INVALID_IDENT;

  begin
    FIRST_COMPONENT_TO_PROCESS := null;
    LAST_COMPONENT_TO_PROCESS  := null;
    NEED_COMMA                 := FALSE;
    loop
      VALID_NEW_COMPONENT_IDENT (TEMP_STRING (1..TEMP_STRING_LAST), WHICH_IDENT);
      case WHICH_IDENT is
        when EOF => PRINT_ERROR ("Incomplete record component - " &
                                 "premature eof found");
                    return FALSE;
        when EOL => PRINT_ERROR ("Incomplete record component - " &
                                 "premature ; found");
                    return FALSE;
        when EOI => if FIRST_COMPONENT_TO_PROCESS = null then
                      PRINT_ERROR ("Incomplete record component -" &
                                   " no valid component identifiers");
                      return FALSE;
                    else
                      return TRUE;
                    end if;
        when COMMA => if not NEED_COMMA then
                        PRINT_ERROR ("Invalid record component - extra comma");
                      end if;
                      NEED_COMMA := FALSE;
        when INVALID_IDENT => null;
```

```
      when VALID_IDENT => if NEED_COMMA then
                            PRINT_ERROR ("Invalid record component " &
                                         "- missing comma");
                          end if;
                          NEED_COMMA := TRUE;
                          NEW_COMPONENT := GET_NEW_COMPONENT_TO_PROCESS_LIST;
                          NEW_COMPONENT.COMPONENT := GET_NEW_LIST_COMPONENT
                                       (TEMP_STRING (1..TEMP_STRING_LAST));
                          ADD_COMPONENT_TO_PROCESS_LIST (NEW_COMPONENT);
    end case;
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
  end loop;
end MAKE_LIST_OF_COMPONENTS;

----------------------------------------------------------------------------
--
-- ADD_COMPONENT_TO_PROCESS_LIST
--
-- if this is the first component-to-process set the first pointer
-- otherwise set the "next" pointer in the previously last
-- component-to-process to point to this new component-to-process
-- set the previous pointer in this new component-to-process to point to the
--           old last component-to-process
-- and now the new component-to-process is the last one

  procedure ADD_COMPONENT_TO_PROCESS_LIST
          (NEW_COMPONENT_TO_PROCESS_LIST : in out
                       ACCESS_COMPONENT_TO_PROCESS_LIST) is
  begin
    if LAST_COMPONENT_TO_PROCESS = null then
      FIRST_COMPONENT_TO_PROCESS := NEW_COMPONENT_TO_PROCESS_LIST;
    else
      LAST_COMPONENT_TO_PROCESS.NEXT_COMPONENT :=
                                NEW_COMPONENT_TO_PROCESS_LIST;
    end if;
    NEW_COMPONENT_TO_PROCESS_LIST.PREVIOUS_COMPONENT :=
                                LAST_COMPONENT_TO_PROCESS;
    LAST_COMPONENT_TO_PROCESS := NEW_COMPONENT_TO_PROCESS_LIST;
  end ADD_COMPONENT_TO_PROCESS_LIST;

----------------------------------------------------------------------------
--
-- GET_NEW_LIST_COMPONENT
--

  function GET_NEW_LIST_COMPONENT
          (TEMP : in STRING)
           return LIST_COMPONENT is
  begin
```

```
    return new LIST_COMPONENT_STRING' (LIST_COMPONENT_STRING (TEMP));
  end GET_NEW_LIST_COMPONENT;

-----------------------------------------------------------------------------
--
-- GET_NEW_COMPONENT_TO_PROCESS_LIST
--

  function GET_NEW_COMPONENT_TO_PROCESS_LIST
           return ACCESS_COMPONENT_TO_PROCESS_LIST is
  begin
    return new COMPONENT_TO_PROCESS_LIST'
       (COMPONENT              => null,
        PREVIOUS_COMPONENT     => null,
        NEXT_COMPONENT         => null);
  end GET_NEW_COMPONENT_TO_PROCESS_LIST;

------------------------------------------------------------------
--
-- MAKE_LIST_OF_VARIABLES
--
-- on entry we should point to a variable name
-- we want to chain up a list of them to process later
-- stop when we find : or ;
-- temp string will contain a variable name on entry
-- they must not contain _NOT_NULL or _NOT_NULL_UNIQUE suffixes
-- they must be unique

  function MAKE_LIST_OF_VARIABLES
           return BOOLEAN is

    NEW_NAME           : ACCESS_NAME_TO_PROCESS_LIST := null;
    NEED_COMMA         : BOOLEAN := FALSE;
    WHICH_IDENT        : IDENT_TYPE := INVALID_IDENT;

  begin
    FIRST_NAME_TO_PROCESS := null;
    LAST_NAME_TO_PROCESS  := null;
    NEED_COMMA            := FALSE;
    loop
      VALID_NEW_VARIABLE_IDENT (TEMP_STRING (1..TEMP_STRING_LAST), WHICH_IDENT);
      case WHICH_IDENT is
        when EOF => return FALSE;
        when EOL => return FALSE;
        when EOI => if FIRST_NAME_TO_PROCESS = null then
                      return FALSE;
                    else
                      return TRUE;
                    end if;
```

```
        when COMMA => if not NEED_COMMA then
                        PRINT_ERROR ("Invalid variable declaration - " &
                                     "extra comma");
                      end if;
                      NEED_COMMA := FALSE;
        when INVALID_IDENT => null;
        when VALID_IDENT => if NEED_COMMA then
                              PRINT_ERROR ("Invalid variable declaration " &
                                           "- missing comma");
                            end if;
                            NEED_COMMA := TRUE;
                            NEW_NAME := GET_NEW_NAME_TO_PROCESS_LIST;
                            NEW_NAME.NAME := GET_NEW_LIST_NAME
                                       (TEMP_STRING (1..TEMP_STRING_LAST));
                            ADD_NAME_TO_PROCESS_LIST (NEW_NAME);
      end case;
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    end loop;
  end MAKE_LIST_OF_VARIABLES;

end LIST_ROUTINES;
```

3.11.120 package ddl_integer_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES;

package INTEGER_ROUTINES is

  procedure PROCESS_INTEGER;

  procedure GET_INTEGER_RANGE
          (VALID              : in out BOOLEAN;
           RANGE_LO           : in out INT;
           RANGE_HI           : in out INT);

  procedure BUILD_INTEGER_TYPE_DESCRIPTORS
          (RANGE_LO         : in INT;
           RANGE_HI         : in INT;
           COUNT            : in out INTEGER);

end INTEGER_ROUTINES;
```

3.11.121 package ddl_integer.ada

```
package body INTEGER_ROUTINES is
```

```
-----------------------------------------------------------------
--
-- PROCESS_INTEGER
--
-- on entry "range" is in temp_string
-- we have to process the statement and determine if it's valid
-- the next token should be an integer for index range lo
-- followed by .. and then an integer for index range hi and then a semi colon

  procedure PROCESS_INTEGER is

    RANGE_LO          : INT := 0;
    RANGE_HI          : INT := 0;
    VALID             : BOOLEAN := TRUE;
    COUNT             : NATURAL := 0;
  begin

-- validate it and store necessary info to build it later

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_INTEGER_RANGE (VALID, RANGE_LO, RANGE_HI);
    if DEBUGGING then
      PRINT_TO_FILE ("    integer range - valid: " & BOOLEAN'IMAGE(VALID) &
                     " range lo: " & INT'IMAGE(RANGE_LO) & " hi: " &
                     INT'IMAGE(RANGE_HI));
    end if;
    if not VALID then
      PRINT_ERROR ("Invalid type - integer declaration, unable to " &
                   " determine range");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    GET_CONSTANT (VALID, ";", FALSE);
    if not VALID then
      PRINT_ERROR ("Invalid type - integer declaration, ending ; missing");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- build type descriptors here

    BUILD_INTEGER_TYPE_DESCRIPTORS (RANGE_LO, RANGE_HI, COUNT);
    if COUNT < 1 then
      PRINT_ERROR ("Invalid type - integer declaration, no valid identifier");
    end if;
    if DEBUGGING then
      PRINT_TO_FILE ("    number of integer type descriptors: " &
                     INTEGER'IMAGE(COUNT));
    end if;
```

```
  end PROCESS_INTEGER;

--------------------------------------------------------------------------
--
-- GET_INTEGER_RANGE
--
-- if valid is false on entry then don't do anything
-- we have to find a range or valid becomes false
-- lo and hi range become the range specified,

  procedure GET_INTEGER_RANGE
           (VALID              : in out BOOLEAN;
            RANGE_LO           : in out INT;
            RANGE_HI           : in out INT) is

    RANGE1 : INT := 0;
    RANGE2 : INT := 0;
    OK     : BOOLEAN := FALSE;

  begin
    if VALID then
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE1);
      if not OK then
        VALID := FALSE;
      else
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        GET_CONSTANT (VALID, ".", TRUE);
        if VALID then
          GET_CONSTANT (VALID, ".", TRUE);
          if VALID then
            STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE2);
            if not OK then
              VALID := FALSE;
            else
              RANGE_LO := RANGE1;
              RANGE_HI := RANGE2;
              GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
              if RANGE_LO > RANGE_HI then
                VALID := FALSE;
              end if;
            end if;
          end if;
        end if;
      end if;
    end if;
  end GET_INTEGER_RANGE;

--------------------------------------------------------------------------
--
```

```
-- BUILD_INTEGER_TYPE_DESCRIPTORS
--

  procedure BUILD_INTEGER_TYPE_DESCRIPTORS
           (RANGE_LO          : in INT;
            RANGE_HI          : in INT;
            COUNT             : in out INTEGER) is

    NAME        : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    INTEGER_DES : ACCESS_INTEGER_DESCRIPTOR := null;

  begin
    COUNT := 0;
    while NAME /= null loop
      if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
        ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
        INTEGER_DES := GET_NEW_INTEGER_DESCRIPTOR;
        FULL_DES.TYPE_IS := INTEGER_DES;
        INTEGER_DES.TYPE_KIND := A_TYPE;
        INTEGER_DES.WHICH_TYPE := INT_EGER;
        INTEGER_DES.BASE_TYPE := INTEGER_DES;
        INTEGER_DES.ULT_PARENT_TYPE := INTEGER_DES;
        INTEGER_DES.FULL_NAME := FULL_DES;
        INTEGER_DES.RANGE_LO_INT := RANGE_LO;
        INTEGER_DES.RANGE_HI_INT := RANGE_HI;
        ADD_TYPE_DESCRIPTOR (INTEGER_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_INTEGER_TYPE_DESCRIPTORS;

end INTEGER_ROUTINES;
```

3.11.122 package ddl_float_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES;

package FLOAT_ROUTINES is
```

```
  procedure PROCESS_FLOAT;

  procedure GET_FLOAT_DIGITS
           (VALID              : in out BOOLEAN;
            DIGIT_INT          : in out INT);

  procedure GET_FLOAT_RANGE
           (VALID              : in out BOOLEAN;
            RANGE_LO           : in out DOUBLE_PRECISION;
            RANGE_HI           : in out DOUBLE_PRECISION);

  procedure BUILD_FLOAT_TYPE_DESCRIPTORS
           (DIGIT_INT        : in INT;
            RANGE_LO         : in DOUBLE_PRECISION;
            RANGE_HI         : in DOUBLE_PRECISION;
            COUNT            : in out INTEGER);

end FLOAT_ROUTINES;
```

3.11.123 package ddl_float.ada

```
package body FLOAT_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_FLOAT
--
-- on entry "digits" is in temp_string
-- we have to process the statement and determine if it's valid
-- the next token must be a positive integer for digits
-- followed by either RANGE or ; -- if RANGE then
-- the next token must be a floating point number for index range lo
-- followed by .. and then a floating point for index range hi and then
-- a semi colon

  procedure PROCESS_FLOAT is

    RANGE_LO          : DOUBLE_PRECISION := 0.0;
    RANGE_HI          : DOUBLE_PRECISION := 0.0;
    VALID             : BOOLEAN := TRUE;
    COUNT             : NATURAL := 0;
    DIGIT_INT         : INT := 0;
    GOT_RANGE         : BOOLEAN := FALSE;

  begin

-- validate it and store necessary info to build it later

    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_FLOAT_DIGITS (VALID, DIGIT_INT);
```

```
    if not VALID then
      PRINT_ERROR ("Invalid type - float declaration, digits must be " &
                   "expressed as a");
      PRINT_TO_FILE ("                positive integer");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    GET_CONSTANT_MAYBE (VALID, GOT_RANGE, "RANGE", TRUE);
    if GOT_RANGE then
      GET_FLOAT_RANGE (VALID, RANGE_LO, RANGE_HI);
      if DEBUGGING then
        PRINT_TO_FILE ("    float range - valid: " & BOOLEAN'IMAGE(VALID) &
                       " range lo: " & DOUBLE_PRECISION_TO_STRING(RANGE_LO) &
                       " hi: " & DOUBLE_PRECISION_TO_STRING(RANGE_HI));
      end if;
      if not VALID then
        PRINT_ERROR ("Invalid type - float declaration, unable to " &
                     " determine range");
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
    end if;
    GET_CONSTANT (VALID, ";", FALSE);
    if not VALID then
      PRINT_ERROR ("Invalid type - float declaration, ending ; missing");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;

-- build type descriptors here

    BUILD_FLOAT_TYPE_DESCRIPTORS (DIGIT_INT, RANGE_LO, RANGE_HI, COUNT);
    if COUNT < 1 then
      PRINT_ERROR ("Invalid type - float declaration, no valid identifier");
    end if;
    if DEBUGGING then
      PRINT_TO_FILE ("    number of float type descriptors: " &
                     NATURAL'IMAGE (COUNT));
    end if;
  end PROCESS_FLOAT;

------------------------------------------------------------------------
--
-- GET_FLOAT_DIGITS
--
-- if valid is false on entry then don't do anything
-- we have to find the float digits which must be a positive integer

  procedure GET_FLOAT_DIGITS
```

```
          (VALID              : in out BOOLEAN;
           DIGIT_INT          : in out INT) is

    D_INT  : INT := 0;
    OK     : BOOLEAN := FALSE;

  begin
    if VALID then
      STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, D_INT);
      if not OK then
        VALID := FALSE;
      else
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        if D_INT < 1 then
          VALID := FALSE;
        else
          DIGIT_INT := D_INT;
        end if;
      end if;
    end if;
  end GET_FLOAT_DIGITS;

------------------------------------------------------------------------
--
-- GET_FLOAT_RANGE
--
-- if valid is false on entry then don't do anything
-- we have to find a range or valid becomes false
-- lo and hi range become the range specified,

  procedure GET_FLOAT_RANGE
          (VALID              : in out BOOLEAN;
           RANGE_LO           : in out DOUBLE_PRECISION;
           RANGE_HI           : in out DOUBLE_PRECISION) is

    RANGE1 : DOUBLE_PRECISION := 0.0;
    RANGE2 : DOUBLE_PRECISION := 0.0;
    OK     : BOOLEAN := FALSE;

  begin
    if VALID then
      STRING_TO_DOUBLE_PRECISION (TEMP_STRING (1..TEMP_STRING_LAST),
                                  OK, RANGE1);
      if not OK then
        VALID := FALSE;
      else
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        GET_CONSTANT (VALID, ".", TRUE);
        if VALID then
```

```
          GET_CONSTANT (VALID, ".", TRUE);
          if VALID then
            STRING_TO_DOUBLE_PRECISION (TEMP_STRING (1..TEMP_STRING_LAST),
                                        OK, RANGE2);
            if not OK then
              VALID := FALSE;
            else
              RANGE_LO := RANGE1;
              RANGE_HI := RANGE2;
              GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
              if RANGE_LO > RANGE_HI then
                VALID := FALSE;
              end if;
            end if;
          end if;
        end if;
      end if;
    end if;
  end GET_FLOAT_RANGE;

----------------------------------------------------------------------------
--
-- BUILD_FLOAT_TYPE_DESCRIPTORS
--

  procedure BUILD_FLOAT_TYPE_DESCRIPTORS
           (DIGIT_INT        : in INT;
            RANGE_LO         : in DOUBLE_PRECISION;
            RANGE_HI         : in DOUBLE_PRECISION;
            COUNT            : in out INTEGER) is

    NAME       : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES  : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES   : ACCESS_FULL_NAME_DESCRIPTOR := null;
    FLOAT_DES  : ACCESS_FLOAT_DESCRIPTOR := null;

  begin
    COUNT := 0;
    while NAME /= null loop
      if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
        ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
        FLOAT_DES := GET_NEW_FLOAT_DESCRIPTOR;
        FULL_DES.TYPE_IS := FLOAT_DES;
        FLOAT_DES.TYPE_KIND := A_TYPE;
        FLOAT_DES.WHICH_TYPE := FL_OAT;
        FLOAT_DES.FULL_NAME := FULL_DES;
        FLOAT_DES.BASE_TYPE := FLOAT_DES;
```

```
        FLOAT_DES.ULT_PARENT_TYPE := FLOAT_DES;
        FLOAT_DES.FLOAT_DIGITS := NATURAL (DIGIT_INT);
        FLOAT_DES.RANGE_LO_FLT := RANGE_LO;
        FLOAT_DES.RANGE_HI_FLT := RANGE_HI;
        ADD_TYPE_DESCRIPTOR (FLOAT_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_FLOAT_TYPE_DESCRIPTORS;

end FLOAT_ROUTINES;
```

3.11.124 package ddl_enumeration_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES,
     SUBROUTINES_4_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, NAME_ROUTINES,
     SUBROUTINES_4_ROUTINES;

package ENUMERATION_ROUTINES is

  procedure PROCESS_ENUMERATION;

  procedure GET_ENUMERATION_LITERAL
           (LIT      : in out STRING;
            LIT_LAST : in out NATURAL);

  function VALID_ENUMERATION_LITERAL
          (LIT_STRING : in STRING)
           return BOOLEAN;

  function DUPLICATE_ENUMERATION_LITERAL
          (ENUM_FIRST    : in ACCESS_LITERAL_DESCRIPTOR;
           ENUM_LAST     : in ACCESS_LITERAL_DESCRIPTOR;
           LIT_STRING    : in STRING)
           return BOOLEAN;

  procedure BUILD_ENUMERATION_TYPE_DESCRIPTORS
           (ENUM_FIRST  : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_LAST   : in ACCESS_LITERAL_DESCRIPTOR;
            POS         : in NATURAL;
            MAX_LEN     : in NATURAL;
            COUNT       : in out NATURAL);
```

```
  procedure BUILD_ENUMERATION_LITERAL_DESCRIPTORS
           (ENUMERATION_DES   : in out ACCESS_ENUMERATION_DESCRIPTOR;
            ENUM_FIRST        : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_LAST         : in ACCESS_LITERAL_DESCRIPTOR);

end ENUMERATION_ROUTINES;
```

3.11.125 package ddl_enumeration.ada

```
package body ENUMERATION_ROUTINES is

------------------------------------------------------------------
--
-- PROCESS_ENUMERATION
--
-- on entry "(" is in temp_string
-- we have to process the statement and determine if it's valid
-- we read enumeration literals up to the next ) or ;

  procedure PROCESS_ENUMERATION is

    VALID             : BOOLEAN := TRUE;
    ERROR             : BOOLEAN := FALSE;
    POS               : NATURAL := 0;
    LIT               : STRING (1..250) := (others => ' ');
    LEN               : NATURAL := 0;
    ENUM_FIRST        : ACCESS_LITERAL_DESCRIPTOR := null;
    ENUM_LAST         : ACCESS_LITERAL_DESCRIPTOR := null;
    ENUM_LIT          : ACCESS_LITERAL_DESCRIPTOR := null;
    MAX_LEN           : NATURAL := 0;
    LIT_STRING        : STRING (1..250) := (others => ' ');
    LIT_LAST          : NATURAL := 0;
    COUNT             : NATURAL := 0;
    GOT_IT            : BOOLEAN := FALSE;

  begin

-- validate it and store necessary info to build it later
    loop
      GET_ENUMERATION_LITERAL (LIT_STRING, LIT_LAST);
      exit when LIT_STRING (1..LIT_LAST) = ";";
      exit when LIT_STRING (1..LIT_LAST) = ")";
      POS := POS + 1;
      if LIT_LAST > MAX_LEN then
        MAX_LEN := LIT_LAST;
      end if;
      if not VALID_ENUMERATION_LITERAL (LIT_STRING (1..LIT_LAST)) then
        ERROR := TRUE;
        PRINT_ERROR ("Invalid enumeration literal: " &
                     LIT_STRING (1..LIT_LAST));
```

```
        else
          if DUPLICATE_ENUMERATION_LITERAL (ENUM_FIRST, ENUM_LAST,
                   LIT_STRING (1..LIT_LAST)) then
            ERROR := TRUE;
            PRINT_TO_FILE ("Duplicate enumeration literal: " &
                           LIT_STRING (1..LIT_LAST));
          else
            ENUM_LIT := GET_NEW_LITERAL_DESCRIPTOR;
            ENUM_LIT.NAME := GET_NEW_ENUMERATION_NAME
                             (LIT_STRING (1..LIT_LAST));
            ENUM_LIT.POS := POS;
            if ENUM_FIRST = null then
              ENUM_FIRST := ENUM_LIT;
            else
              ENUM_LAST.NEXT_LITERAL := ENUM_LIT;
            end if;
            ENUM_LIT.PREVIOUS_LITERAL := ENUM_LAST;
            ENUM_LAST := ENUM_LIT;
          end if;
        end if;
      end loop;
      GET_CONSTANT_MAYBE (VALID, GOT_IT, ")", TRUE);
      VALID := TRUE;
      GET_CONSTANT (VALID, ";", FALSE);
      if not VALID then
        PRINT_ERROR ("Invalid type - enumeration declaration, ending ; missing");
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
        return;
      end if;
      if ERROR then
        PRINT_ERROR ("Invalid type - enumeration declaration ignored");
        return;
      end if;

-- build type descriptors here

      BUILD_ENUMERATION_TYPE_DESCRIPTORS (ENUM_FIRST, ENUM_LAST, POS, MAX_LEN,
                         COUNT);
      if COUNT < 1 then
        PRINT_ERROR ("Invalid type - enumeration declaration, " &
                     "no valid identifier");
      end if;
      if DEBUGGING then
        PRINT_TO_FILE ("    number of enumeration type descriptors: " &
                       INTEGER'IMAGE(COUNT));
      end if;
    end PROCESS_ENUMERATION;

-----------------------------------------------------------------------
```

```
--
-- GET_ENUMERATION_LITERAL
--
-- enumeration literals my be an identifier or a single character in a quote
-- if the first character read is a quote read until another quote
-- if the second is a quote then read for another quote

  procedure GET_ENUMERATION_LITERAL
           (LIT      : in out STRING;
            LIT_LAST : in out NATURAL) is

    TEMP_LAST : NATURAL := 0;
    VALID     : BOOLEAN := FALSE;

  begin
    LIT_LAST := 0;
    LIT (1) := ' ';
    if TEMP_STRING (1..TEMP_STRING_LAST) = ";" or
       TEMP_STRING (1..TEMP_STRING_LAST) = ")" then
      LIT_LAST := TEMP_STRING_LAST;
      LIT (1..LIT_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
      return;
    end if;
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    LIT_LAST := TEMP_STRING_LAST;
    LIT (1..LIT_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
    VALID := TRUE;
    if TEMP_STRING (1..TEMP_STRING_LAST) = "'" then
      GET_SINGLE_QUOTE_STRING (CURRENT_SCHEMA_UNIT, LIT, LIT_LAST, VALID);
    end if;
    if VALID then
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    end if;
    loop
      exit when TEMP_STRING (1..TEMP_STRING_LAST) = ",";
      exit when TEMP_STRING (1..TEMP_STRING_LAST) = ")";
      exit when TEMP_STRING (1..TEMP_STRING_LAST) = ";";
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      PRINT_ERROR ("Invalid separator in enumeration literal list: " &
                   TEMP_STRING (1..TEMP_STRING_LAST));
    end loop;
  end GET_ENUMERATION_LITERAL;

-----------------------------------------------------------------------
--
-- VALID_ENUMERATION_LITERAL
--
-- valid enumeration literals are either valid identifiers or a single
-- character between single quotes
```

```
  function VALID_ENUMERATION_LITERAL
          (LIT_STRING : in STRING)
           return BOOLEAN is
  begin
    if LIT_STRING (LIT_STRING'FIRST) /= ''' then
      return VALID_IDENT_CHARS (LIT_STRING);
    else
      return ((LIT_STRING'LAST - LIT_STRING'FIRST) = 2) and
             LIT_STRING (LIT_STRING'FIRST) = ''' and
             LIT_STRING (LIT_STRING'LAST) = ''';
    end if;
  end VALID_ENUMERATION_LITERAL;

-------------------------------------------------------------------------
--
-- DUPLICATE_ENUMERATION_LITERAL
--
-- see if this literal has been used before in this enumeration

  function DUPLICATE_ENUMERATION_LITERAL
          (ENUM_FIRST    : in ACCESS_LITERAL_DESCRIPTOR;
           ENUM_LAST     : in ACCESS_LITERAL_DESCRIPTOR;
           LIT_STRING    : in STRING)
           return BOOLEAN is

    ENUM : ACCESS_LITERAL_DESCRIPTOR := ENUM_FIRST;

  begin
    while ENUM /= null loop
      if LIT_STRING = STRING (ENUM.NAME.all) then
        return TRUE;
      end if;
      ENUM := ENUM.NEXT_LITERAL;
    end loop;
    return FALSE;
  end DUPLICATE_ENUMERATION_LITERAL;


-------------------------------------------------------------------------
--
-- BUILD_ENUMERATION_TYPE_DESCRIPTORS
--

  procedure BUILD_ENUMERATION_TYPE_DESCRIPTORS
           (ENUM_FIRST  : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_LAST   : in ACCESS_LITERAL_DESCRIPTOR;
            POS         : in NATURAL;
            MAX_LEN     : in NATURAL;
            COUNT       : in out NATURAL) is
```

```
    NAME             : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES        : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES         : ACCESS_FULL_NAME_DESCRIPTOR := null;
    ENUMERATION_DES  : ACCESS_ENUMERATION_DESCRIPTOR := null;

  begin
    COUNT := 0;
    while NAME /= null loop
      if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
        ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
        ENUMERATION_DES := GET_NEW_ENUMERATION_DESCRIPTOR;
        FULL_DES.TYPE_IS := ENUMERATION_DES;
        ENUMERATION_DES.TYPE_KIND := A_TYPE;
        ENUMERATION_DES.WHICH_TYPE := ENUMERATION;
        ENUMERATION_DES.FULL_NAME := FULL_DES;
        ENUMERATION_DES.BASE_TYPE := ENUMERATION_DES;
        ENUMERATION_DES.ULT_PARENT_TYPE := ENUMERATION_DES;
        ENUMERATION_DES.LAST_POS := POS;
        ENUMERATION_DES.MAX_LENGTH := MAX_LEN;
        BUILD_ENUMERATION_LITERAL_DESCRIPTORS (ENUMERATION_DES, ENUM_FIRST,
                     ENUM_LAST);
        ADD_TYPE_DESCRIPTOR (ENUMERATION_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_ENUMERATION_TYPE_DESCRIPTORS;
--------------------------------------------------------------------------
--
-- BUILD_ENUMERATION_LITERAL_DESCRIPTORS
--
  procedure BUILD_ENUMERATION_LITERAL_DESCRIPTORS
           (ENUMERATION_DES  : in out ACCESS_ENUMERATION_DESCRIPTOR;
            ENUM_FIRST       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_LAST        : in ACCESS_LITERAL_DESCRIPTOR) is

    ADD_THIS_LITERAL : ACCESS_LITERAL_DESCRIPTOR := ENUM_FIRST;
    NEW_LITERAL      : ACCESS_LITERAL_DESCRIPTOR := null;

  begin
    while ADD_THIS_LITERAL /= null loop
      NEW_LITERAL             := GET_NEW_LITERAL_DESCRIPTOR;
      NEW_LITERAL.NAME        := ADD_THIS_LITERAL.NAME;
      NEW_LITERAL.POS         := ADD_THIS_LITERAL.POS;
      NEW_LITERAL.PARENT_ENUM := ENUMERATION_DES;
```

```
      ADD_LITERAL_DESCRIPTOR (NEW_LITERAL, ENUMERATION_DES);
      ADD_NEW_ENUM_LIT (ENUMERATION_DES, STRING (ADD_THIS_LITERAL.NAME.all));
      ADD_THIS_LITERAL := ADD_THIS_LITERAL.NEXT_LITERAL;
    end loop;
  end BUILD_ENUMERATION_LITERAL_DESCRIPTORS;

end ENUMERATION_ROUTINES;
```

3.11.126 package ddl_derived_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, SUBROUTINES_3_ROUTINES,
     SUBROUTINES_4_ROUTINES, NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, SUBROUTINES_2_ROUTINES, SUBROUTINES_3_ROUTINES,
     SUBROUTINES_4_ROUTINES, NAME_ROUTINES;

package DERIVED_ROUTINES is

  procedure PROCESS_DERIVED;

  procedure BUILD_DERIVED_TYPE_DESCRIPTORS
           (COUNT               : in out NATURAL;
            PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in BOOLEAN;
            ARRAY_INDEX_LO      : in INT;
            ARRAY_INDEX_HI      : in INT;
            GOT_INTEGER_RANGE   : in BOOLEAN;
            INTEGER_RANGE_LO    : in INT;
            INTEGER_RANGE_HI    : in INT;
            GOT_FLOAT_DIGITS    : in BOOLEAN;
            FLOAT_DIGITS        : in NATURAL;
            GOT_FLOAT_RANGE     : in BOOLEAN;
            FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE      : in BOOLEAN;
            ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS            : in NATURAL);

end DERIVED_ROUTINES;
```

3.11.127 package ddl_derived.ada

```
package body DERIVED_ROUTINES is

-----------------------------------------------------------------
--
```

```
-- PROCESS_DERIVED
--
-- on entry "new" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a derived type descriptor

  procedure PROCESS_DERIVED is
    VALID                : BOOLEAN :=  TRUE;
    ERROR_NUMBER         : NATURAL :=  0;
    PARENT_DES           : ACCESS_TYPE_DESCRIPTOR :=  null;
    GOT_ARRAY_INDEX      : BOOLEAN :=  FALSE;
    ARRAY_INDEX_LO       : INT :=  0;
    ARRAY_INDEX_HI       : INT :=  0;
    GOT_INTEGER_RANGE    : BOOLEAN :=  FALSE;
    INTEGER_RANGE_LO     : INT :=  0;
    INTEGER_RANGE_HI     : INT :=  0;
    GOT_FLOAT_DIGITS     : BOOLEAN :=  FALSE;
    FLOAT_DIGITS         : NATURAL :=  0;
    GOT_FLOAT_RANGE      : BOOLEAN :=  FALSE;
    FLOAT_RANGE_LO       : DOUBLE_PRECISION :=  0.0;
    FLOAT_RANGE_HI       : DOUBLE_PRECISION :=  0.0;
    GOT_ENUM_RANGE       : BOOLEAN :=  FALSE;
    ENUM_RANGE_LO        : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_RANGE_HI        : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_POS             : NATURAL :=  0;
    COUNT                : NATURAL := 0;

  begin
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    BREAK_DOWN_SUBTYPE_INDICATOR (VALID, ERROR_NUMBER, PARENT_DES,
              GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI,
              GOT_INTEGER_RANGE, INTEGER_RANGE_LO, INTEGER_RANGE_HI,
              GOT_FLOAT_DIGITS, FLOAT_DIGITS, GOT_FLOAT_RANGE,
              FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE, ENUM_RANGE_LO,
              ENUM_RANGE_HI, ENUM_POS);
    if DEBUGGING then
      PRINT_TO_FILE ("    break down subtype indicator");
      PRINT_TO_FILE ("    valid: " & BOOLEAN'IMAGE(VALID) & " error number: "
                     & INTEGER'IMAGE (ERROR_NUMBER));
      if PARENT_DES /= null then
         PRINT_TO_FILE ("    parent: " &
         STRING (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
         STRING (PARENT_DES.FULL_NAME.NAME.all));
      else
         PRINT_TO_FILE ("    parent: descriptor null");
      end if;
      PRINT_TO_FILE ("    got_array_index: " &
                     BOOLEAN'IMAGE (GOT_ARRAY_INDEX) & " array index lo: " &
                     INT'IMAGE (ARRAY_INDEX_LO) & " array index hi: " &
```

```
                  INT'IMAGE (ARRAY_INDEX_HI));
    PRINT_TO_FILE ("    got integer range: " &
                  BOOLEAN'IMAGE (GOT_INTEGER_RANGE) & " integer range lo: "
                  & INT'IMAGE (INTEGER_RANGE_LO) & " integer range hi: " &
                  INT'IMAGE (INTEGER_RANGE_HI));
    PRINT_TO_FILE ("    got float digits: " &
                  BOOLEAN'IMAGE (GOT_FLOAT_DIGITS) & " float digits: " &
                  INTEGER'IMAGE (FLOAT_DIGITS));
    PRINT_TO_FILE ("    got float range: " &
                  BOOLEAN'IMAGE (GOT_FLOAT_RANGE) & " float range lo: " &
                  DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_LO) &
                  " float range hi: " &
                  DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_HI));
    PRINT_TO_FILE ("    got enum range: " & BOOLEAN'IMAGE (GOT_ENUM_RANGE));
    if ENUM_RANGE_LO /= null then
      PRINT_TO_FILE ("    enum range lo: " & STRING (ENUM_RANGE_LO.NAME.all));
    end if;
    if ENUM_RANGE_HI /= null then
      PRINT_TO_FILE ("    enum range hi: " & STRING (ENUM_RANGE_HI.NAME.all));
    end if;
    PRINT_TO_FILE ("    enum pos: " & INTEGER'IMAGE (ENUM_POS));
  end if;
  if VALID then
    BUILD_DERIVED_TYPE_DESCRIPTORS (COUNT, PARENT_DES, GOT_ARRAY_INDEX,
              ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
              INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS,
              FLOAT_DIGITS, GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI,
              GOT_ENUM_RANGE, ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
    if DEBUGGING then
      PRINT_TO_FILE ("    build derived type descriptors - count: " &
                    INTEGER'IMAGE (COUNT));
    end if;
    if COUNT < 1 then
      PRINT_ERROR ("Invalid derived descriptor - identifier not valid");
    end if;
  else
    PRINT_ERROR ("Invalid derived declaration - subtype indicator invalid");
    case ERROR_NUMBER is
      when  1 => PRINT_TO_FILE ("    identifier invalid");
      when  2 => PRINT_TO_FILE ("    identifier is a component");
      when  3 => PRINT_TO_FILE ("    identifier is a record");
      when  4 => PRINT_TO_FILE ("    invalid enumeration range");
      when  5 => PRINT_TO_FILE
                 ("    invalid enumeration range literals");
      when  6 => PRINT_TO_FILE ("    invalid range for integer");
      when  7 => PRINT_TO_FILE ("    invalid range for integer");
      when  8 => PRINT_TO_FILE ("    invalid digits or range for float");
      when  9 => PRINT_TO_FILE ("    invalid digits for float");
      when 10 => PRINT_TO_FILE ("    invalid range for float");
```

```
        when 11 => PRINT_TO_FILE ("    invalid range for string");
        when 12 => PRINT_TO_FILE ("    invalid range for string");
        when 13 => PRINT_TO_FILE
                   ("    range was given for a constrained array");
        when 14 => PRINT_TO_FILE
                   ("    range was not given for an unconstrained array");
        when others => PRINT_TO_FILE ("    unknown error");
      end case;
    end if;
    if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
      PRINT_ERROR ("Invalid derived descriptor - no ending ; found");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
    end if;
  end PROCESS_DERIVED;

------------------------------------------------------------------------
--
-- BUILD_DERIVED_TYPE_DESCRIPTORS
--

  procedure BUILD_DERIVED_TYPE_DESCRIPTORS
           (COUNT              : in out NATURAL;
            PARENT_DES         : in ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX    : in BOOLEAN;
            ARRAY_INDEX_LO     : in INT;
            ARRAY_INDEX_HI     : in INT;
            GOT_INTEGER_RANGE  : in BOOLEAN;
            INTEGER_RANGE_LO   : in INT;
            INTEGER_RANGE_HI   : in INT;
            GOT_FLOAT_DIGITS   : in BOOLEAN;
            FLOAT_DIGITS       : in NATURAL;
            GOT_FLOAT_RANGE    : in BOOLEAN;
            FLOAT_RANGE_LO     : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI     : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE     : in BOOLEAN;
            ENUM_RANGE_LO      : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI      : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS           : in NATURAL) is

    NAME        : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    DERIVED_DES : ACCESS_TYPE_DESCRIPTOR := null;

  begin
    COUNT := 0;
    while NAME /= null loop
      if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
```

```
        ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
        DERIVED_DES := GET_NEW_TYPE_DESCRIPTOR (PARENT_DES.WHICH_TYPE);
        FULL_DES.TYPE_IS := DERIVED_DES;
        DERIVED_DES.TYPE_KIND := A_DERIVED;
        DERIVED_DES.FULL_NAME := FULL_DES;
        INSERT_SUBTYPE_INDICATOR_INFORMATION (PARENT_DES, DERIVED_DES,
           GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
           INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS, FLOAT_DIGITS,
           GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE,
           ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
        ADD_TYPE_DESCRIPTOR (DERIVED_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
      end if;
      if DERIVED_DES.WHICH_TYPE = ENUMERATION then
        ADD_NEW_ENUM_LIT_FOR_DERIVED (DERIVED_DES);
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_DERIVED_TYPE_DESCRIPTORS;

end DERIVED_ROUTINES;
```

3.11.128 package ddl_variable_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, GET_NEW_DESCRIPTOR_ROUTINES, ERROR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, GET_NEW_DESCRIPTOR_ROUTINES, ERROR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;

package VARIABLE_ROUTINES is

  procedure TRY_TO_PROCESS_VARIABLE;

  procedure PROCESS_VARIABLE;

  procedure BUILD_VARIABLE_TYPE_DESCRIPTORS
           (COUNT               : in out NATURAL;
            PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in BOOLEAN;
            ARRAY_INDEX_LO      : in INT;
            ARRAY_INDEX_HI      : in INT;
            GOT_INTEGER_RANGE   : in BOOLEAN;
            INTEGER_RANGE_LO    : in INT;
```

```
            INTEGER_RANGE_HI     : in INT;
            GOT_FLOAT_DIGITS     : in BOOLEAN;
            FLOAT_DIGITS         : in NATURAL;
            GOT_FLOAT_RANGE      : in BOOLEAN;
            FLOAT_RANGE_LO       : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI       : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE       : in BOOLEAN;
            ENUM_RANGE_LO        : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI        : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS             : in NATURAL);

end VARIABLE_ROUTINES;
```

3.11.129 package ddl_variable.ada

```
package body VARIABLE_ROUTINES is

------------------------------------------------------------------
--
-- TRY_TO_PROCESS_VARIABLE
--
-- first thing to do is store away the identifier or identifiers
-- if there are identifiers and then a : we assume variables, otherwise
-- we assume it's a statement we know nothing about
-- then process the subtype indicator then build it all into a variable descripto

  procedure TRY_TO_PROCESS_VARIABLE is
  begin

-- first make chain of all identifiers returns with ":" in temp_string
    if DEBUGGING then
      PRINT_TO_FILE ("*** VARIABLE");
    end if;
    if MAKE_LIST_OF_VARIABLES then
      if CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES or
         CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES or
         CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE or
         CURRENT_SCHEMA_UNIT.AUTH_ID /= null or
         CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE = null or
         (CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /=
          CURRENT_SCHEMA_UNIT.LAST_DECLARED_PACKAGE) or
         (CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE /= null and
          CURRENT_SCHEMA_UNIT.FIRST_DECLARED_PACKAGE.FOUND_END) then
        PRINT_ERROR ("Program variables for use with Ada/SQL must " &
                     "stand alone in a compilation unit");
        PRINT_TO_FILE ("   within a single non-nested package.  Types, " &
                       "tables, and authorization");
        PRINT_TO_FILE ("   statements are not permitted in a variable " &
                       "package.  This variable");
        PRINT_TO_FILE ("   declaration will be ignored.");
```

```
      else
        CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES := TRUE;
        PROCESS_VARIABLE;
      end if;
    else
      PROCESS_ERROR;
    end if;
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
  end TRY_TO_PROCESS_VARIABLE;

-----------------------------------------------------------------
--
-- PROCESS_VARIABLE
--
-- on entry ":" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a variable type descriptor

  procedure PROCESS_VARIABLE is
    VALID                  : BOOLEAN :=  TRUE;
    ERROR_NUMBER           : NATURAL :=  0;
    PARENT_DES             : ACCESS_TYPE_DESCRIPTOR :=  null;
    GOT_ARRAY_INDEX        : BOOLEAN :=  FALSE;
    ARRAY_INDEX_LO         : INT :=  0;
    ARRAY_INDEX_HI         : INT :=  0;
    GOT_INTEGER_RANGE      : BOOLEAN :=  FALSE;
    INTEGER_RANGE_LO       : INT :=  0;
    INTEGER_RANGE_HI       : INT :=  0;
    GOT_FLOAT_DIGITS       : BOOLEAN :=  FALSE;
    FLOAT_DIGITS           : NATURAL :=  0;
    GOT_FLOAT_RANGE        : BOOLEAN :=  FALSE;
    FLOAT_RANGE_LO         : DOUBLE_PRECISION :=  0.0;
    FLOAT_RANGE_HI         : DOUBLE_PRECISION :=  0.0;
    GOT_ENUM_RANGE         : BOOLEAN :=  FALSE;
    ENUM_RANGE_LO          : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_RANGE_HI          : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_POS               : NATURAL :=  0;
    COUNT                  : NATURAL := 0;

  begin
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    BREAK_DOWN_SUBTYPE_INDICATOR (VALID, ERROR_NUMBER, PARENT_DES,
              GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI,
              GOT_INTEGER_RANGE, INTEGER_RANGE_LO, INTEGER_RANGE_HI,
              GOT_FLOAT_DIGITS, FLOAT_DIGITS, GOT_FLOAT_RANGE,
              FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE, ENUM_RANGE_LO,
              ENUM_RANGE_HI, ENUM_POS);
    if DEBUGGING then
      PRINT_TO_FILE ("    break down subtype indicator");
```

```
      PRINT_TO_FILE ("    valid: " & BOOLEAN'IMAGE(VALID) &
         " error number: " & INTEGER'IMAGE (ERROR_NUMBER));
      if (PARENT_DES = null or else
         PARENT_DES.FULL_NAME = null or else
         (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME = null and
         PARENT_DES.FULL_NAME.NAME = null)) then
        PRINT_TO_FILE ("    parent: null.null");
      elsif PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME = null then
        PRINT_TO_FILE ("    parent: null." &
           STRING (PARENT_DES.FULL_NAME.NAME.all));
      elsif PARENT_DES.FULL_NAME.NAME = null then
        PRINT_TO_FILE ("    parent: " &
           STRING (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME.all) & ".null");
      else
        PRINT_TO_FILE ("    parent: " &
           STRING (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
           STRING (PARENT_DES.FULL_NAME.NAME.all));
      end if;
      PRINT_TO_FILE ("    got_array_index: " &
         BOOLEAN'IMAGE (GOT_ARRAY_INDEX) & " array index lo: " &
         INT'IMAGE (ARRAY_INDEX_LO) & " array index hi: " &
         INT'IMAGE (ARRAY_INDEX_HI));
      PRINT_TO_FILE ("    got integer range: " &
          BOOLEAN'IMAGE (GOT_INTEGER_RANGE) & " integer range lo: " &
          INT'IMAGE (INTEGER_RANGE_LO) & " integer range hi: " &
          INT'IMAGE (INTEGER_RANGE_HI));
      PRINT_TO_FILE ("    got float digits: " &
          BOOLEAN'IMAGE (GOT_FLOAT_DIGITS) & " float digits: " &
          INTEGER'IMAGE (FLOAT_DIGITS));
      PRINT_TO_FILE ("    got float range: " & BOOLEAN'IMAGE (GOT_FLOAT_RANGE)
          & " float range lo: " & DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_LO)
          & " float ragne hi: " & DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_HI));
      PRINT_TO_FILE ("    got enum range: " & BOOLEAN'IMAGE (GOT_ENUM_RANGE));
      if ENUM_RANGE_LO /= null then
        PRINT_TO_FILE ("    enum range lo: " & STRING (ENUM_RANGE_LO.NAME.all));
      end if;
      if ENUM_RANGE_HI /= null then
        PRINT_TO_FILE ("    enum range hi: " & STRING (ENUM_RANGE_HI.NAME.all));
      end if;
      PRINT_TO_FILE ("    enum pos: " & INTEGER'IMAGE (ENUM_POS));
    end if;
    if VALID then
      BUILD_VARIABLE_TYPE_DESCRIPTORS (COUNT, PARENT_DES, GOT_ARRAY_INDEX,
                ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
                INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS,
                FLOAT_DIGITS, GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI,
                GOT_ENUM_RANGE, ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
      if DEBUGGING then
        PRINT_TO_FILE ("    build variable type descriptors - count: " &
```

```
                          INTEGER'IMAGE (COUNT));
        end if;
        if COUNT < 1 then
          PRINT_ERROR ("Invalid variable declaration - identifier not valid");
        end if;
      else
        PRINT_ERROR ("Invalid variable declaration - subtype indicator invalid");
        case ERROR_NUMBER is
          when  1 => PRINT_TO_FILE ("    identifier invalid");
          when  2 => PRINT_TO_FILE ("    identifier is a component");
          when  3 => PRINT_TO_FILE ("    identifier is a record");
          when  4 => PRINT_TO_FILE ("    invalid enumeration range");
          when  5 => PRINT_TO_FILE
                     ("    invalid enumeration range literals");
          when  6 => PRINT_TO_FILE ("    invalid range for integer");
          when  7 => PRINT_TO_FILE ("    invalid range for integer");
          when  8 => PRINT_TO_FILE ("    invalid digits or range for float");
          when  9 => PRINT_TO_FILE ("    invalid digits for float");
          when 10 => PRINT_TO_FILE ("    invalid range for float");
          when 11 => PRINT_TO_FILE ("    invalid range for string");
          when 12 => PRINT_TO_FILE ("    invalid range for string");
          when 13 => PRINT_TO_FILE
                     ("    range was given for a constrained array");
          when 14 => PRINT_TO_FILE
                     ("    range was not given for an unconstrained array");
          when others => PRINT_TO_FILE ("    unknown error");
        end case;
      end if;
      if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        PRINT_ERROR ("Invalid subtype descriptor - no ending ; found");
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      end if;
    end PROCESS_VARIABLE;

--------------------------------------------------------------------------
--
-- BUILD_VARIABLE_TYPE_DESCRIPTORS
--

    procedure BUILD_VARIABLE_TYPE_DESCRIPTORS
             (COUNT              : in out NATURAL;
              PARENT_DES         : in ACCESS_TYPE_DESCRIPTOR;
              GOT_ARRAY_INDEX    : in BOOLEAN;
              ARRAY_INDEX_LO     : in INT;
              ARRAY_INDEX_HI     : in INT;
              GOT_INTEGER_RANGE  : in BOOLEAN;
              INTEGER_RANGE_LO   : in INT;
              INTEGER_RANGE_HI   : in INT;
              GOT_FLOAT_DIGITS   : in BOOLEAN;
```

```
        FLOAT_DIGITS        : in NATURAL;
        GOT_FLOAT_RANGE     : in BOOLEAN;
        FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
        FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
        GOT_ENUM_RANGE      : in BOOLEAN;
        ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
        ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
        ENUM_POS            : in NATURAL) is

  NAME         : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
  IDENT_DES    : ACCESS_IDENTIFIER_DESCRIPTOR := null;
  FULL_DES     : ACCESS_FULL_NAME_DESCRIPTOR := null;
  VARIABLE_DES : ACCESS_TYPE_DESCRIPTOR := null;
  OK           : BOOLEAN := TRUE;
  NULL_UNIQUE  : BOOLEAN := FALSE;
  IS_NULL      : BOOLEAN := FALSE;
  IS_UNIQUE    : BOOLEAN := FALSE;

begin
  COUNT := 0;
  while NAME /= null loop
    if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
      IS_IDENTIFIER_NULL_OR_UNIQUE (STRING (NAME.NAME.all), IS_NULL,
                    IS_UNIQUE);
      IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
      ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
         (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
      VARIABLE_DES := GET_NEW_TYPE_DESCRIPTOR (PARENT_DES.WHICH_TYPE);
      FULL_DES.TYPE_IS := VARIABLE_DES;
      VARIABLE_DES.TYPE_KIND := A_VARIABLE;
      VARIABLE_DES.FULL_NAME := FULL_DES;
      INSERT_SUBTYPE_INDICATOR_INFORMATION (PARENT_DES, VARIABLE_DES,
         GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
         INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS, FLOAT_DIGITS,
         GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE,
         ENUM_RANGE_LO, ENUM_RANGE_HI,ENUM_POS);
      VARIABLE_DES.NOT_NULL := IS_NULL;
      VARIABLE_DES.NOT_NULL_UNIQUE := IS_UNIQUE;
      ADD_VARIABLE_TYPE_DESCRIPTOR (VARIABLE_DES);
      COUNT := COUNT + 1;
    else
      PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
    end if;
    VALIDATE_NULL_UNIQUE_CONSTRAINTS (VARIABLE_DES, PARENT_DES,
          NULL_UNIQUE, OK);
    if NULL_UNIQUE and (GOT_ARRAY_INDEX or GOT_INTEGER_RANGE or
         GOT_FLOAT_DIGITS or GOT_FLOAT_RANGE or GOT_ENUM_RANGE) then
      PRINT_ERROR ("Variables with null/unique constraints cannot provide " &
                   "subtype indicator");
```

```
        PRINT_TO_FILE ("        constraints");
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_VARIABLE_TYPE_DESCRIPTORS;

end VARIABLE_ROUTINES;
```

3.11.130 package ddl_subtype_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, GET_NEW_DESCRIPTOR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, GET_NEW_DESCRIPTOR_ROUTINES,
     ADD_DESCRIPTOR_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;

package SUBTYPE_ROUTINES is

  procedure PROCESS_SUBTYPE;

  procedure DO_A_SUBTYPE;

  procedure BUILD_SUBTYPE_TYPE_DESCRIPTORS
           (COUNT                : in out NATURAL;
            PARENT_DES           : in ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX      : in BOOLEAN;
            ARRAY_INDEX_LO       : in INT;
            ARRAY_INDEX_HI       : in INT;
            GOT_INTEGER_RANGE    : in BOOLEAN;
            INTEGER_RANGE_LO     : in INT;
            INTEGER_RANGE_HI     : in INT;
            GOT_FLOAT_DIGITS     : in BOOLEAN;
            FLOAT_DIGITS         : in NATURAL;
            GOT_FLOAT_RANGE      : in BOOLEAN;
            FLOAT_RANGE_LO       : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI       : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE       : in BOOLEAN;
            ENUM_RANGE_LO        : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI        : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS             : in NATURAL);

end SUBTYPE_ROUTINES;
```

3.11.131 package ddl_subtype.ada

```
package body SUBTYPE_ROUTINES is
```

```
-----------------------------------------------------------------
--
-- PROCESS_SUBTYPE
--
-- first thing to do is store away the identifier or identifiers
-- then process the subtype indicator then build it all into a type descriptor

  procedure PROCESS_SUBTYPE is
  begin

-- first make chain of all identifiers returns with "is" in temp_string
    if DEBUGGING then
      PRINT_TO_FILE ("*** SUBTYPE");
    end if;
    CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES := TRUE;
    if CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE then
      PRINT_ERROR ("Subtype declarations are not permitted within " &
                   "an authorization package");
    end if;
    if CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES then
      PRINT_ERROR ("Subtype declarations must not be declared in a " &
                   "compilation unit which also");
      PRINT_TO_FILE ("   declares Ada/SQL program variables");
    end if;
    if not IN_ADA_SQL_PACKAGE then
      PRINT_ERROR ("Subtype declarations permitted only within the ADA_SQL " &
                   " subpackages");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    if MAKE_LIST_OF_NAMES then
      DO_A_SUBTYPE;
    end if;
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
  end PROCESS_SUBTYPE;

-----------------------------------------------------------------
--
-- DO_A_SUBTYPE
--
-- on entry "is" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a subtype type descriptor

  procedure DO_A_SUBTYPE is
    VALID              : BOOLEAN :=  TRUE;
    ERROR_NUMBER       : NATURAL :=  0;
    PARENT_DES         : ACCESS_TYPE_DESCRIPTOR :=  null;
    GOT_ARRAY_INDEX    : BOOLEAN :=  FALSE;
```

```
    ARRAY_INDEX_LO      : INT :=  0;
    ARRAY_INDEX_HI      : INT :=  0;
    GOT_INTEGER_RANGE   : BOOLEAN :=  FALSE;
    INTEGER_RANGE_LO    : INT :=  0;
    INTEGER_RANGE_HI    : INT :=  0;
    GOT_FLOAT_DIGITS    : BOOLEAN :=  FALSE;
    FLOAT_DIGITS        : NATURAL :=  0;
    GOT_FLOAT_RANGE     : BOOLEAN :=  FALSE;
    FLOAT_RANGE_LO      : DOUBLE_PRECISION :=  0.0;
    FLOAT_RANGE_HI      : DOUBLE_PRECISION :=  0.0;
    GOT_ENUM_RANGE      : BOOLEAN :=  FALSE;
    ENUM_RANGE_LO       : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_RANGE_HI       : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_POS            : NATURAL :=  0;
    COUNT               : NATURAL := 0;

  begin
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    BREAK_DOWN_SUBTYPE_INDICATOR (VALID, ERROR_NUMBER, PARENT_DES,
              GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI,
              GOT_INTEGER_RANGE, INTEGER_RANGE_LO, INTEGER_RANGE_HI,
              GOT_FLOAT_DIGITS, FLOAT_DIGITS, GOT_FLOAT_RANGE,
              FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE, ENUM_RANGE_LO,
              ENUM_RANGE_HI, ENUM_POS);
    if DEBUGGING then
      PRINT_TO_FILE ("    break down subtype indicator");
      PRINT_TO_FILE ("    valid: " & BOOLEAN'IMAGE(VALID) &
         " error number: " & INTEGER'IMAGE (ERROR_NUMBER));
--------------------------------------------------------------------------
      PRINT_TO_FILE ("in subtype");
      if PARENT_DES = null then
        PRINT_TO_FILE ("parent_des is null");
      else
        PRINT_TO_FILE ("parent_des is not null");
        if PARENT_DES.FULL_NAME = null then
          PRINT_TO_FILE ("parent_des.full_name is null");
        else
          PRINT_TO_FILE ("parent_des.full_name is not null");
          if PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME = null then
            PRINT_TO_FILE ("parent_des.full_name.full_package_name is null");
          else
            PRINT_TO_FILE ("parent_des.full_name.full_package_name is not null");
          end if;
          if PARENT_DES.FULL_NAME.NAME = null then
            PRINT_TO_FILE ("parent_des.full_name.name is null");
          else
            PRINT_TO_FILE ("parent_des.full_name.name is not null");
          end if;
        end if;
```

```
    end if;
--------------------------------------------------------------------------
    PRINT_TO_FILE ("    parent: " &
       STRING (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
       STRING (PARENT_DES.FULL_NAME.NAME.all));
    PRINT_TO_FILE ("    got_array_index: " &
       BOOLEAN'IMAGE (GOT_ARRAY_INDEX) & " array index lo: " &
       INT'IMAGE (ARRAY_INDEX_LO) & " array index hi: " &
       INT'IMAGE (ARRAY_INDEX_HI));
    PRINT_TO_FILE ("    got integer range: " &
        BOOLEAN'IMAGE (GOT_INTEGER_RANGE) & " integer range lo: " &
        INT'IMAGE (INTEGER_RANGE_LO) & " integer range hi: " &
        INT'IMAGE (INTEGER_RANGE_HI));
    PRINT_TO_FILE ("    got float digits: " &
        BOOLEAN'IMAGE (GOT_FLOAT_DIGITS) & " float digits: " &
        INTEGER'IMAGE (FLOAT_DIGITS));
    PRINT_TO_FILE ("    got float range: " & BOOLEAN'IMAGE (GOT_FLOAT_RANGE)
        & " float range lo: " & DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_LO)
        & " float ragne hi: " & DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_HI));
    PRINT_TO_FILE ("    got enum range: " & BOOLEAN'IMAGE (GOT_ENUM_RANGE));
    if ENUM_RANGE_LO /= null then
      PRINT_TO_FILE ("    enum range lo: " & STRING (ENUM_RANGE_LO.NAME.all));
    end if;
    if ENUM_RANGE_HI /= null then
      PRINT_TO_FILE ("    enum range hi: " & STRING (ENUM_RANGE_HI.NAME.all));
    end if;
    PRINT_TO_FILE ("    enum pos: " & INTEGER'IMAGE (ENUM_POS));
  end if;
  if VALID then
    BUILD_SUBTYPE_TYPE_DESCRIPTORS (COUNT, PARENT_DES, GOT_ARRAY_INDEX,
              ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
              INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS,
              FLOAT_DIGITS, GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI,
              GOT_ENUM_RANGE, ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
    if DEBUGGING then
      PRINT_TO_FILE ("    build subtype type descriptors - count: " &
                     INTEGER'IMAGE (COUNT));
    end if;
    if COUNT < 1 then
      PRINT_ERROR ("Invalid subtype descriptor - identifier not valid");
    end if;
  else
    PRINT_ERROR ("Invalid subtype declaration - subtype indicator invalid");
    case ERROR_NUMBER is
      when  1 => PRINT_TO_FILE ("    identifier invalid");
      when  2 => PRINT_TO_FILE ("    identifier is a component");
      when  3 => PRINT_TO_FILE ("    identifier is a record");
      when  4 => PRINT_TO_FILE ("    invalid enumeration range");
      when  5 => PRINT_TO_FILE
```

```
                ("    invalid enumeration range literals");
      when  6 => PRINT_TO_FILE ("    invalid range for integer");
      when  7 => PRINT_TO_FILE ("    invalid range for integer");
      when  8 => PRINT_TO_FILE ("    invalid digits or range for float");
      when  9 => PRINT_TO_FILE ("    invalid digits for float");
      when 10 => PRINT_TO_FILE ("    invalid range for float");
      when 11 => PRINT_TO_FILE ("    invalid range for string");
      when 12 => PRINT_TO_FILE ("    invalid range for string");
      when 13 => PRINT_TO_FILE
                ("    range was given for a constrained array");
      when 14 => PRINT_TO_FILE
                ("    range was not given for an unconstrained array");
      when others => PRINT_TO_FILE ("    unknown error");
    end case;
  end if;
  if not GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
    PRINT_ERROR ("Invalid subtype descriptor - no ending ; found");
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
  end if;
end DO_A_SUBTYPE;

------------------------------------------------------------------------
--
-- BUILD_SUBTYPE_TYPE_DESCRIPTORS
--

  procedure BUILD_SUBTYPE_TYPE_DESCRIPTORS
          (COUNT                : in out NATURAL;
           PARENT_DES           : in ACCESS_TYPE_DESCRIPTOR;
           GOT_ARRAY_INDEX      : in BOOLEAN;
           ARRAY_INDEX_LO       : in INT;
           ARRAY_INDEX_HI       : in INT;
           GOT_INTEGER_RANGE    : in BOOLEAN;
           INTEGER_RANGE_LO     : in INT;
           INTEGER_RANGE_HI     : in INT;
           GOT_FLOAT_DIGITS     : in BOOLEAN;
           FLOAT_DIGITS         : in NATURAL;
           GOT_FLOAT_RANGE      : in BOOLEAN;
           FLOAT_RANGE_LO       : in DOUBLE_PRECISION;
           FLOAT_RANGE_HI       : in DOUBLE_PRECISION;
           GOT_ENUM_RANGE       : in BOOLEAN;
           ENUM_RANGE_LO        : in ACCESS_LITERAL_DESCRIPTOR;
           ENUM_RANGE_HI        : in ACCESS_LITERAL_DESCRIPTOR;
           ENUM_POS             : in NATURAL) is

    NAME        : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    SUBTYPE_DES : ACCESS_TYPE_DESCRIPTOR := null;
```

```
    OK          : BOOLEAN := TRUE;
    NULL_UNIQUE : BOOLEAN := FALSE;
    IS_NULL     : BOOLEAN := FALSE;
    IS_UNIQUE   : BOOLEAN := FALSE;

  begin
    COUNT := 0;
    while NAME /= null loop
      if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
        IS_IDENTIFIER_NULL_OR_UNIQUE (STRING (NAME.NAME.all), IS_NULL,
                      IS_UNIQUE);
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
        ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
        SUBTYPE_DES := GET_NEW_TYPE_DESCRIPTOR (PARENT_DES.WHICH_TYPE);
        FULL_DES.TYPE_IS := SUBTYPE_DES;
        SUBTYPE_DES.TYPE_KIND := A_SUBTYPE;
        SUBTYPE_DES.FULL_NAME := FULL_DES;
        INSERT_SUBTYPE_INDICATOR_INFORMATION (PARENT_DES, SUBTYPE_DES,
           GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
           INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS, FLOAT_DIGITS,
           GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE,
           ENUM_RANGE_LO, ENUM_RANGE_HI,ENUM_POS);
        SUBTYPE_DES.NOT_NULL := IS_NULL;
        SUBTYPE_DES.NOT_NULL_UNIQUE := IS_UNIQUE;
        ADD_TYPE_DESCRIPTOR (SUBTYPE_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
      end if;
      VALIDATE_NULL_UNIQUE_CONSTRAINTS (SUBTYPE_DES, PARENT_DES,
            NULL_UNIQUE, OK);
      if NULL_UNIQUE and (GOT_ARRAY_INDEX or GOT_INTEGER_RANGE or
           GOT_FLOAT_DIGITS or GOT_FLOAT_RANGE or GOT_ENUM_RANGE) then
        PRINT_ERROR ("Subtype with null/unique constraints cannot provide " &
                     "subtype indicator");
        PRINT_TO_FILE ("        constraints");
      end if;
      NAME := NAME.NEXT_NAME;
    end loop;
  end BUILD_SUBTYPE_TYPE_DESCRIPTORS;

end SUBTYPE_ROUTINES;
```

3.11.132 package ddl_record_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;
```

```
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     SUBROUTINES_3_ROUTINES, SUBROUTINES_4_ROUTINES, NAME_ROUTINES;

package RECORD_ROUTINES is

  procedure PROCESS_RECORD;

  procedure BUILD_COMPONENT_TYPE_DESCRIPTORS
           (FIRST_COMPONENT     : in out ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            LAST_COMPONENT      : in out ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            PREV_COUNT          : in out NATURAL;
            NOW_COUNT           : in out NATURAL;
            PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
            GOT_ARRAY_INDEX     : in BOOLEAN;
            ARRAY_INDEX_LO      : in INT;
            ARRAY_INDEX_HI      : in INT;
            GOT_INTEGER_RANGE   : in BOOLEAN;
            INTEGER_RANGE_LO    : in INT;
            INTEGER_RANGE_HI    : in INT;
            GOT_FLOAT_DIGITS    : in BOOLEAN;
            FLOAT_DIGITS        : in NATURAL;
            GOT_FLOAT_RANGE     : in BOOLEAN;
            FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
            FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
            GOT_ENUM_RANGE      : in BOOLEAN;
            ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS            : in NATURAL);

  procedure BUILD_RECORD_TYPE_DESCRIPTORS
           (FIRST_COMPONENT     : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            LAST_COMPONENT      : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            COUNT               : in out NATURAL);

  procedure INSERT_COMPONENT_DESCRIPTORS
           (RECORD_DES        : in out ACCESS_RECORD_DESCRIPTOR;
            COMPONENT_FIRST   : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            COMPONENT_LAST    : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            COUNT             : in out NATURAL);

end RECORD_ROUTINES;
```

3.11.133 package ddl_record.ada

```
package body RECORD_ROUTINES is

-----------------------------------------------------------------
--
```

```
-- PROCESS_RECORD
--
-- on entry "record" is in temp_string
-- we have to process each component statement and determine if it's valid
-- read token to get first component name or "end", if end we're done with
-- the whole record, if component name call make_list_of_components to
-- stack up the component names since there may be more than one.

  procedure PROCESS_RECORD is
    VALID                  : BOOLEAN :=  TRUE;
    ERROR_NUMBER           : NATURAL :=  0;
    PARENT_DES             : ACCESS_TYPE_DESCRIPTOR :=  null;
    GOT_ARRAY_INDEX        : BOOLEAN :=  FALSE;
    ARRAY_INDEX_LO         : INT :=  0;
    ARRAY_INDEX_HI         : INT :=  0;
    GOT_INTEGER_RANGE      : BOOLEAN :=  FALSE;
    INTEGER_RANGE_LO       : INT :=  0;
    INTEGER_RANGE_HI       : INT :=  0;
    GOT_FLOAT_DIGITS       : BOOLEAN :=  FALSE;
    FLOAT_DIGITS           : NATURAL :=  0;
    GOT_FLOAT_RANGE        : BOOLEAN :=  FALSE;
    FLOAT_RANGE_LO         : DOUBLE_PRECISION :=  0.0;
    FLOAT_RANGE_HI         : DOUBLE_PRECISION :=  0.0;
    GOT_ENUM_RANGE         : BOOLEAN :=  FALSE;
    ENUM_RANGE_LO          : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_RANGE_HI          : ACCESS_LITERAL_DESCRIPTOR :=  null;
    ENUM_POS               : NATURAL :=  0;
    FIRST_COMPONENT        : ACCESS_HOLDING_COMPONENT_DESCRIPTOR := null;
    LAST_COMPONENT         : ACCESS_HOLDING_COMPONENT_DESCRIPTOR := null;
    PREV_COUNT             : NATURAL := 0;
    NOW_COUNT              : NATURAL := 0;
    COUNT                  : NATURAL := 0;

  begin
    CURRENT_SCHEMA_UNIT.HAS_DECLARED_TABLES := TRUE;
    if CURRENT_SCHEMA_UNIT.AUTH_ID = null and then
       STRING (CURRENT_SCHEMA_UNIT.NAME.all) /= CURSOR_NAME then
      PRINT_ERROR ("Records (tables) must be declared in a schema " &
                   "unit with an associated");
      PRINT_TO_FILE ("authorization identifier");
    end if;
    loop
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      exit when TEMP_STRING(1..TEMP_STRING_LAST) = "END";
      exit when CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE;
      if MAKE_LIST_OF_COMPONENTS then
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        BREAK_DOWN_SUBTYPE_INDICATOR (VALID, ERROR_NUMBER, PARENT_DES,
              GOT_ARRAY_INDEX, ARRAY_INDEX_LO, ARRAY_INDEX_HI,
```

```
      GOT_INTEGER_RANGE, INTEGER_RANGE_LO, INTEGER_RANGE_HI,
      GOT_FLOAT_DIGITS, FLOAT_DIGITS, GOT_FLOAT_RANGE,
      FLOAT_RANGE_LO, FLOAT_RANGE_HI, GOT_ENUM_RANGE, ENUM_RANGE_LO,
      ENUM_RANGE_HI, ENUM_POS);
if VALID and then PARENT_DES.WHICH_TYPE = STR_ING and then
   (not PARENT_DES.CONSTRAINED) and then (not GOT_ARRAY_INDEX) then
  VALID := FALSE;
  ERROR_NUMBER := 14;
end if;
if DEBUGGING then
  PRINT_TO_FILE ("            break down subtype indicator");
  PRINT_TO_FILE ("            valid: " & BOOLEAN'IMAGE(VALID) &
     " error number: " & INTEGER'IMAGE (ERROR_NUMBER));
  if PARENT_DES /= null then
    PRINT_TO_FILE ("            parent: " &
     STRING (PARENT_DES.FULL_NAME.FULL_PACKAGE_NAME.all) & "." &
     STRING (PARENT_DES.FULL_NAME.NAME.all));
  else
     PRINT_TO_FILE ("            parent: null");
  end if;
  PRINT_TO_FILE ("            got_array_index: " &
     BOOLEAN'IMAGE (GOT_ARRAY_INDEX) & " array index lo: " &
     INT'IMAGE (ARRAY_INDEX_LO) & " array index hi: " &
     INT'IMAGE (ARRAY_INDEX_HI));
  PRINT_TO_FILE ("            got integer range: " &
      BOOLEAN'IMAGE (GOT_INTEGER_RANGE) & " integer range lo: " &
      INT'IMAGE (INTEGER_RANGE_LO) & " integer range hi: " &
      INT'IMAGE (INTEGER_RANGE_HI));
  PRINT_TO_FILE ("            got float digits: " &
      BOOLEAN'IMAGE (GOT_FLOAT_DIGITS) & " float digits: " &
      INTEGER'IMAGE (FLOAT_DIGITS));
  PRINT_TO_FILE ("           got float range: " &
      BOOLEAN'IMAGE (GOT_FLOAT_RANGE) & " float range lo: " &
      DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_LO) &
      " float range hi: " &
      DOUBLE_PRECISION_TO_STRING (FLOAT_RANGE_HI));
  PRINT_TO_FILE ("            got enum range: " &
      BOOLEAN'IMAGE (GOT_ENUM_RANGE));
  if ENUM_RANGE_LO /= null then
    PRINT_TO_FILE ("            enum range lo: " &
      STRING (ENUM_RANGE_LO.NAME.all));
  end if;
  if ENUM_RANGE_HI /= null then
    PRINT_TO_FILE ("            enum range hi: " &
      STRING (ENUM_RANGE_HI.NAME.all));
  end if;
  PRINT_TO_FILE ("           enum pos: " &
      INTEGER'IMAGE (ENUM_POS));
end if;
```

```
        if VALID then
          BUILD_COMPONENT_TYPE_DESCRIPTORS (FIRST_COMPONENT, LAST_COMPONENT,
                PREV_COUNT, NOW_COUNT, PARENT_DES, GOT_ARRAY_INDEX,
                ARRAY_INDEX_LO, ARRAY_INDEX_HI, GOT_INTEGER_RANGE,
                INTEGER_RANGE_LO, INTEGER_RANGE_HI, GOT_FLOAT_DIGITS,
                FLOAT_DIGITS, GOT_FLOAT_RANGE, FLOAT_RANGE_LO, FLOAT_RANGE_HI,
                GOT_ENUM_RANGE, ENUM_RANGE_LO, ENUM_RANGE_HI, ENUM_POS);
          if DEBUGGING then
            PRINT_TO_FILE ("            build component type descriptors -" &
                   " count prev: " & INTEGER'IMAGE (PREV_COUNT) & " now: " &
                   INTEGER'IMAGE (NOW_COUNT));
          end if;
          if NOW_COUNT <= PREV_COUNT or NOW_COUNT < 1 then
            PRINT_ERROR ("Invalid record descriptor, no component " &
                          "identifiers declared");
          end if;
          PREV_COUNT := NOW_COUNT;
          NOW_COUNT := 0;
        else
          PRINT_ERROR ("Invalid record declaration - component's " &
                        "subtype indicator was invalid");
          case ERROR_NUMBER is
            when  1 => PRINT_TO_FILE ("    identifier invalid");
            when  2 => PRINT_TO_FILE ("    identifier is a component");
            when  3 => PRINT_TO_FILE ("    identifier is a record");
            when  4 => PRINT_TO_FILE ("    invalid enumeration range");
            when  5 => PRINT_TO_FILE
                        ("    invalid enumeration range literals");
            when  6 => PRINT_TO_FILE ("    invalid range for integer");
            when  7 => PRINT_TO_FILE ("    invalid range for integer");
            when  8 => PRINT_TO_FILE ("    invalid digits or range for float");
            when  9 => PRINT_TO_FILE ("    invalid digits for float");
            when 10 => PRINT_TO_FILE ("    invalid range for float");
            when 11 => PRINT_TO_FILE ("    invalid range for string");
            when 12 => PRINT_TO_FILE ("    invalid range for string");
            when 13 => PRINT_TO_FILE
                        ("    range was given for a constrained array");
            when 14 => PRINT_TO_FILE
                        ("    range was not given for an unconstrained array");
            when others => PRINT_TO_FILE ("    unknown error");
          end case;
        end if;
      else
        PRINT_ERROR ("Invalid record descriptor, no component " &
                      "identifiers declared");
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      end if;
    end loop;
    if TEMP_STRING (1..TEMP_STRING_LAST) = "END" then
```

```
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if (TEMP_STRING (1..TEMP_STRING_LAST)) = "RECORD" then
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if GOT_END_OF_STATEMENT (TEMP_STRING (1..TEMP_STRING_LAST)) then
        if PREV_COUNT > 0 then
          BUILD_RECORD_TYPE_DESCRIPTORS (FIRST_COMPONENT, LAST_COMPONENT,
                   COUNT);
          if DEBUGGING then
            PRINT_TO_FILE ("           build record type descriptors -" &
                     " count: " & INTEGER'IMAGE (COUNT));
          end if;
          if COUNT < 1 then
          PRINT_ERROR ("Invalid record descriptor, no identifiers" &
                       " declared");
          end if;
        else
          PRINT_ERROR ("Invalid record descriptor - no components declared");
        end if;
        return;
      end if;
    end if;
  end if;
  PRINT_ERROR ("Invalid record descriptor");
  FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
end PROCESS_RECORD;

-------------------------------------------------------------------------
--
-- BUILD_COMPONENT_TYPE_DESCRIPTORS
--

  procedure BUILD_COMPONENT_TYPE_DESCRIPTORS
          (FIRST_COMPONENT     : in out ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
           LAST_COMPONENT      : in out ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
           PREV_COUNT          : in out NATURAL;
           NOW_COUNT           : in out NATURAL;
           PARENT_DES          : in ACCESS_TYPE_DESCRIPTOR;
           GOT_ARRAY_INDEX     : in BOOLEAN;
           ARRAY_INDEX_LO      : in INT;
           ARRAY_INDEX_HI      : in INT;
           GOT_INTEGER_RANGE   : in BOOLEAN;
           INTEGER_RANGE_LO    : in INT;
           INTEGER_RANGE_HI    : in INT;
           GOT_FLOAT_DIGITS    : in BOOLEAN;
           FLOAT_DIGITS        : in NATURAL;
           GOT_FLOAT_RANGE     : in BOOLEAN;
           FLOAT_RANGE_LO      : in DOUBLE_PRECISION;
           FLOAT_RANGE_HI      : in DOUBLE_PRECISION;
           GOT_ENUM_RANGE      : in BOOLEAN;
```

```
            ENUM_RANGE_LO       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_RANGE_HI       : in ACCESS_LITERAL_DESCRIPTOR;
            ENUM_POS            : in NATURAL) is

  COMPONENT        : ACCESS_COMPONENT_TO_PROCESS_LIST :=
                        FIRST_COMPONENT_TO_PROCESS;
  COMPONENT_DES    : ACCESS_HOLDING_COMPONENT_DESCRIPTOR := null;
  OK               : BOOLEAN := TRUE;

begin
  NOW_COUNT := PREV_COUNT;
  while COMPONENT /= null loop
    if VALID_NEW_IDENT_NAME_DUPS_OK (STRING (COMPONENT.COMPONENT.all)) then
      COMPONENT_DES := new HOLDING_COMPONENT_DESCRIPTOR;
      COMPONENT_DES.WHICH_TYPE  := PARENT_DES.WHICH_TYPE;
      COMPONENT_DES.FULL_NAME   := GET_NEW_TYPE_NAME
                                   (STRING (COMPONENT.COMPONENT.all));
      COMPONENT_DES.PARENT_TYPE := PARENT_DES;
      COMPONENT_DES.BASE_TYPE := PARENT_DES.BASE_TYPE;
      COMPONENT_DES.ULT_PARENT_TYPE := PARENT_DES.ULT_PARENT_TYPE;
      COMPONENT_DES.NOT_NULL := PARENT_DES.NOT_NULL;
      COMPONENT_DES.NOT_NULL_UNIQUE := PARENT_DES.NOT_NULL_UNIQUE;
      case PARENT_DES.WHICH_TYPE is
        when REC_ORD     => PRINT_ERROR ("Invalid record component -" &
                                  " parent cannot be a record type");
                            OK := FALSE;
                            COMPONENT_DES := null;
        when ENUMERATION =>
             if GOT_ENUM_RANGE then
               COMPONENT_DES.FIRST_LITERAL := ENUM_RANGE_LO;
               COMPONENT_DES.LAST_LITERAL  := ENUM_RANGE_HI;
               COMPONENT_DES.LAST_POS      := ENUM_POS;
               COMPONENT_DES.MAX_LENGTH    := PARENT_DES.MAX_LENGTH;
             else
               COMPONENT_DES.FIRST_LITERAL := PARENT_DES.FIRST_LITERAL;
               COMPONENT_DES.LAST_LITERAL  := PARENT_DES.LAST_LITERAL;
               COMPONENT_DES.LAST_POS      := PARENT_DES.LAST_POS;
               COMPONENT_DES.MAX_LENGTH    := PARENT_DES.MAX_LENGTH;
             end if;
        when INT_EGER    =>
             if GOT_INTEGER_RANGE then
               COMPONENT_DES.RANGE_LO_INT := INTEGER_RANGE_LO;
               COMPONENT_DES.RANGE_HI_INT := INTEGER_RANGE_HI;
             else
               COMPONENT_DES.RANGE_LO_INT := PARENT_DES.RANGE_LO_INT;
               COMPONENT_DES.RANGE_HI_INT := PARENT_DES.RANGE_HI_INT;
             end if;
        when FL_OAT      =>
             if GOT_FLOAT_DIGITS then
```

```
              COMPONENT_DES.FLOAT_DIGITS := FLOAT_DIGITS;
            else
              COMPONENT_DES.FLOAT_DIGITS := PARENT_DES.FLOAT_DIGITS;
            end if;
            if GOT_FLOAT_RANGE then
              COMPONENT_DES.RANGE_LO_FLT := FLOAT_RANGE_LO;
              COMPONENT_DES.RANGE_HI_FLT := FLOAT_RANGE_HI;
            else
              COMPONENT_DES.RANGE_LO_FLT := PARENT_DES.RANGE_LO_FLT;
              COMPONENT_DES.RANGE_HI_FLT := PARENT_DES.RANGE_HI_FLT;
            end if;
        when STR_ING     =>
            COMPONENT_DES.INDEX_TYPE := PARENT_DES.INDEX_TYPE;
            COMPONENT_DES.ARRAY_TYPE := PARENT_DES.ARRAY_TYPE;
            COMPONENT_DES.CONSTRAINED := TRUE;
            COMPONENT_DES.ARRAY_RANGE_MIN := PARENT_DES.ARRAY_RANGE_MIN;
            COMPONENT_DES.ARRAY_RANGE_MAX := PARENT_DES.ARRAY_RANGE_MAX;
            if GOT_ARRAY_INDEX then
              COMPONENT_DES.ARRAY_RANGE_LO := ARRAY_INDEX_LO;
              COMPONENT_DES.ARRAY_RANGE_HI := ARRAY_INDEX_HI;
              COMPONENT_DES.LENGTH := INTEGER
                         (ARRAY_INDEX_HI - ARRAY_INDEX_LO + 1);
            else
              COMPONENT_DES.ARRAY_RANGE_LO := PARENT_DES.ARRAY_RANGE_LO;
              COMPONENT_DES.ARRAY_RANGE_HI := PARENT_DES.ARRAY_RANGE_HI;
              COMPONENT_DES.LENGTH := PARENT_DES.LENGTH;
            end if;
      end case;
    else
      PRINT_ERROR ("Invalid identifier: " &
                   STRING (COMPONENT.COMPONENT.all));
    end if;
    if OK then
      NOW_COUNT := NOW_COUNT + 1;
      if FIRST_COMPONENT = null then
        FIRST_COMPONENT := COMPONENT_DES;
      else
        LAST_COMPONENT.NEXT_COMPONENT := COMPONENT_DES;
      end if;
      COMPONENT_DES.PREVIOUS_COMPONENT := LAST_COMPONENT;
      LAST_COMPONENT := COMPONENT_DES;
    end if;
    COMPONENT := COMPONENT.NEXT_COMPONENT;
  end loop;
end BUILD_COMPONENT_TYPE_DESCRIPTORS;

------------------------------------------------------------------------
--
-- BUILD_RECORD_TYPE_DESCRIPTORS
```

```
--

procedure BUILD_RECORD_TYPE_DESCRIPTORS
         (FIRST_COMPONENT    : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
          LAST_COMPONENT     : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
          COUNT              : in out NATURAL) is

  NAME       : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
  IDENT_DES  : ACCESS_IDENTIFIER_DESCRIPTOR := null;
  FULL_DES   : ACCESS_FULL_NAME_DESCRIPTOR := null;
  RECORD_DES : ACCESS_RECORD_DESCRIPTOR := null;
  OK         : BOOLEAN := TRUE;
  COM_COUNT  : NATURAL := 0;
  REC_COUNT  : NATURAL := 0;

begin
  COUNT := 0;
  while NAME /= null loop
    REC_COUNT := REC_COUNT + 1;
    if REC_COUNT > 1 then
      PRINT_ERROR ("Invalid record declaration, atempting to " &
                   "declare multiple records");
      return;
    end if;
    VALID_NEW_TABLE_NAME (STRING (NAME.NAME.all), IDENT_DES, OK);
    if OK then
      ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
         (IDENT_DES, FULL_DES, STRING (NAME.NAME.all));
      RECORD_DES := GET_NEW_RECORD_DESCRIPTOR;
      FULL_DES.TYPE_IS := RECORD_DES;
      RECORD_DES.TYPE_KIND := A_TYPE;
      RECORD_DES.WHICH_TYPE := REC_ORD;
      RECORD_DES.FULL_NAME := FULL_DES;
      RECORD_DES.BASE_TYPE := RECORD_DES;
      RECORD_DES.ULT_PARENT_TYPE := RECORD_DES;
      INSERT_COMPONENT_DESCRIPTORS (RECORD_DES, FIRST_COMPONENT,
                   LAST_COMPONENT, COM_COUNT);
      if COM_COUNT > 0 then
        ADD_RECORD_TYPE_DESCRIPTOR (RECORD_DES);
        COUNT := COUNT + 1;
      else
        PRINT_ERROR ("Invalid record declaration, no components");
      end if;
    else
      PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
    end if;
    NAME := NAME.NEXT_NAME;
  end loop;
end BUILD_RECORD_TYPE_DESCRIPTORS;
```

```
------------------------------------------------------------------------
--
-- INSERT_COMPONENT_DESCRIPTORS
--
  procedure INSERT_COMPONENT_DESCRIPTORS
           (RECORD_DES        : in out ACCESS_RECORD_DESCRIPTOR;
            COMPONENT_FIRST   : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            COMPONENT_LAST    : in ACCESS_HOLDING_COMPONENT_DESCRIPTOR;
            COUNT             : in out NATURAL) is

    ADD_THIS_COMPONENT : ACCESS_HOLDING_COMPONENT_DESCRIPTOR
                         := COMPONENT_FIRST;
    NEW_COMPONENT      : ACCESS_TYPE_DESCRIPTOR := null;
    IDENT_DES          : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES           : ACCESS_FULL_NAME_DESCRIPTOR := null;

  begin
    COUNT := 0;
    while ADD_THIS_COMPONENT /= null loop
      if VALID_NEW_FULL_COMPONENT_NAME
                      (STRING (ADD_THIS_COMPONENT.FULL_NAME.all),
                       STRING (RECORD_DES.FULL_NAME.NAME.all)) then
        IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR
                     (STRING (ADD_THIS_COMPONENT.FULL_NAME.all));
        ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS
           (IDENT_DES, FULL_DES, STRING (ADD_THIS_COMPONENT.FULL_NAME.all),
            STRING (RECORD_DES.FULL_NAME.NAME.all));
        NEW_COMPONENT                    := GET_NEW_TYPE_DESCRIPTOR
                                           (ADD_THIS_COMPONENT.WHICH_TYPE);
        FULL_DES.TYPE_IS                 := NEW_COMPONENT;
        FULL_DES.IS_NOT_NULL             := ADD_THIS_COMPONENT.NOT_NULL;
        FULL_DES.IS_NOT_NULL_UNIQUE      := ADD_THIS_COMPONENT.NOT_NULL_UNIQUE;
        NEW_COMPONENT.TYPE_KIND          := A_COMPONENT;
        NEW_COMPONENT.WHICH_TYPE         := ADD_THIS_COMPONENT.WHICH_TYPE;
        NEW_COMPONENT.NOT_NULL           := ADD_THIS_COMPONENT.NOT_NULL;
        NEW_COMPONENT.NOT_NULL_UNIQUE    := ADD_THIS_COMPONENT.NOT_NULL_UNIQUE;
        NEW_COMPONENT.PARENT_TYPE        := ADD_THIS_COMPONENT.PARENT_TYPE;
        NEW_COMPONENT.BASE_TYPE          := ADD_THIS_COMPONENT.BASE_TYPE;
        NEW_COMPONENT.ULT_PARENT_TYPE    := ADD_THIS_COMPONENT.ULT_PARENT_TYPE;
        NEW_COMPONENT.PARENT_RECORD      := RECORD_DES;
        NEW_COMPONENT.FULL_NAME          := FULL_DES;
        case NEW_COMPONENT.WHICH_TYPE is
          when REC_ORD      => null;
          when ENUMERATION =>
               NEW_COMPONENT.FIRST_LITERAL := ADD_THIS_COMPONENT.FIRST_LITERAL;
               NEW_COMPONENT.LAST_LITERAL  := ADD_THIS_COMPONENT.LAST_LITERAL;
               NEW_COMPONENT.LAST_POS      := ADD_THIS_COMPONENT.LAST_POS;
               NEW_COMPONENT.MAX_LENGTH    := ADD_THIS_COMPONENT.MAX_LENGTH;
          when INT_EGER     =>
```

```
                    NEW_COMPONENT.RANGE_LO_INT := ADD_THIS_COMPONENT.RANGE_LO_INT;
                    NEW_COMPONENT.RANGE_HI_INT := ADD_THIS_COMPONENT.RANGE_HI_INT;
               when FL_OAT      =>
                    NEW_COMPONENT.FLOAT_DIGITS := ADD_THIS_COMPONENT.FLOAT_DIGITS;
                    NEW_COMPONENT.RANGE_LO_FLT := ADD_THIS_COMPONENT.RANGE_LO_FLT;
                    NEW_COMPONENT.RANGE_HI_FLT := ADD_THIS_COMPONENT.RANGE_HI_FLT;
               when STR_ING     =>
                    NEW_COMPONENT.LENGTH          := ADD_THIS_COMPONENT.LENGTH;
                    NEW_COMPONENT.INDEX_TYPE      := ADD_THIS_COMPONENT.INDEX_TYPE;
                    NEW_COMPONENT.ARRAY_TYPE      := ADD_THIS_COMPONENT.ARRAY_TYPE;
                    NEW_COMPONENT.CONSTRAINED     := ADD_THIS_COMPONENT.CONSTRAINED;
                    NEW_COMPONENT.ARRAY_RANGE_LO  :=
                                           ADD_THIS_COMPONENT.ARRAY_RANGE_LO;
                    NEW_COMPONENT.ARRAY_RANGE_HI  :=
                                           ADD_THIS_COMPONENT.ARRAY_RANGE_HI;
                    NEW_COMPONENT.ARRAY_RANGE_MIN :=
                                           ADD_THIS_COMPONENT.ARRAY_RANGE_MIN;
                    NEW_COMPONENT.ARRAY_RANGE_MAX :=
                                           ADD_THIS_COMPONENT.ARRAY_RANGE_MAX;
             end case;
             ADD_TYPE_DESCRIPTOR (NEW_COMPONENT);
             if RECORD_DES.FIRST_COMPONENT = null then
               RECORD_DES.FIRST_COMPONENT := NEW_COMPONENT;
             else
               RECORD_DES.LAST_COMPONENT.NEXT_ONE := NEW_COMPONENT;
             end if;
             NEW_COMPONENT.PREVIOUS_ONE := RECORD_DES.LAST_COMPONENT;
             RECORD_DES.LAST_COMPONENT := NEW_COMPONENT;
             COUNT := COUNT + 1;
           else
             PRINT_ERROR ("Invalid component identifier " &
                          STRING (ADD_THIS_COMPONENT.FULL_NAME.all));
           end if;
             ADD_THIS_COMPONENT := ADD_THIS_COMPONENT.NEXT_COMPONENT;
         end loop;
       end INSERT_COMPONENT_DESCRIPTORS;

     end RECORD_ROUTINES;
```

3.11.134 package ddl_array_spec.ada

```
with DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     NAME_ROUTINES;
use  DATABASE, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SUBROUTINES_1_ROUTINES, LIST_ROUTINES, SUBROUTINES_2_ROUTINES,
     NAME_ROUTINES;
```

```
package ARRAY_ROUTINES is

  procedure PROCESS_ARRAY;

  procedure GET_ARRAY_INDEX_TYPE
           (VALID                 : in out BOOLEAN;
            GOT_INDEX_TYPE        : in out BOOLEAN;
            INDEX_TYPE            : in out STRING;
            INDEX_TYPE_LAST       : in out NATURAL;
            RANGE_MIN             : in out INT;
            RANGE_MAX             : in out INT;
            INDEX_TYPE_DES        : in out ACCESS_TYPE_DESCRIPTOR);

  procedure GET_ARRAY_INDEX_RANGE
           (VALID                : in out BOOLEAN;
            NEED_RANGE           : in BOOLEAN;
            GOT_RANGE            : in out BOOLEAN;
            RANGE_LO             : in out INT;
            RANGE_HI             : in out INT;
            GOT_INDEX_TYPE       : in BOOLEAN;
            RANGE_MIN            : in out INT;
            RANGE_MAX            : in out INT;
            CONSTRAINED          : in out BOOLEAN);

  procedure GET_ARRAY_TYPE_OF
           (VALID              : in out BOOLEAN;
            GOT_ARRAY_TYPE     : in out BOOLEAN;
            ARRAY_TYPE         : in out STRING;
            ARRAY_TYPE_LAST    : in out NATURAL;
            ARRAY_TYPE_DES     : in out ACCESS_TYPE_DESCRIPTOR);

  procedure BUILD_STRING_TYPE_DESCRIPTORS
           (INDEX_TYPE_DES    : in ACCESS_TYPE_DESCRIPTOR;
            ARRAY_TYPE_DES    : in ACCESS_TYPE_DESCRIPTOR;
            CONSTRAINED       : in BOOLEAN;
            RANGE_LO          : in INT;
            RANGE_HI          : in INT;
            RANGE_MIN         : in INT;
            RANGE_MAX         : in INT;
            COUNT             : in out INTEGER);

end ARRAY_ROUTINES;
```

3.11.135 package ddl_array.ada

```
package body ARRAY_ROUTINES is

-----------------------------------------------------------------
--
-- PROCESS_ARRAY
```

```
--
-- on entry "array" is in temp_string
-- we have to process the statement and determine if it's valid
-- an unconstrained array is valid as follows:
--     ( index-type RANGE <> ) OF identifier
-- a constrained array is valid as follows:
--     ( index_type ) OF identifier
--     ( index_type RANGE l..h ) OF identifier
--     ( l..h ) OF identifier
-- if valid we collect the following information about the array to be stored
-- in the type descriptor:
--
-- identifier name    - to create a new identifier descriptor or be included
--                      in an existing one (captured by process_type, stored
--                      in make_list_of_names)
-- full name pointer  - a pointer to a full name descriptor pointed to from
--                      the identifier descriptor
-- string length      - hi range - lo range + 1, unless it's constrained then
--                      use zero for now
-- index type         - a pointer to the type descriptor of the index type,
--                      which must be base type of integer, if one is
--                      specified, if not we use standard.integer at the type
-- array type         - a pointer to the type descriptor of the array type,
--                      which muse be a base type of character
-- constrained        - true if it is false if it isn't
-- index range min    - if index type is supplied we have the minimum possible
--                      for the range, must be >= 0
-- index range max    - if index type is supplied we have the maximum possible
--                      for the range, must be >= 0
-- index range lo     - if an actual range is supplied this is the lo value,
--                      must be >= 0, unless the array is unconstrained then
--                      it will be -1
-- index range hi     - if an actual range is supplied this is the hi value,
--                      must be >= 0, unless the array is unconstrained then
--                      it will be -1


  procedure PROCESS_ARRAY is

    GOT_INDEX_TYPE    : BOOLEAN := FALSE;
    GOT_ARRAY_TYPE    : BOOLEAN := FALSE;
    INDEX_TYPE        : STRING (1..250) := (others => ' ');
    INDEX_TYPE_LAST   : NATURAL := 0;
    ARRAY_TYPE        : STRING (1..250) := (others => ' ');
    ARRAY_TYPE_LAST   : NATURAL := 0;
    NEED_RANGE        : BOOLEAN := FALSE;
    GOT_RANGE         : BOOLEAN := FALSE;
    VALID             : BOOLEAN := TRUE;
    RANGE_LO          : INT := 0;
```

```
    RANGE_HI          : INT := 0;
    RANGE_MIN         : INT := 0;
    RANGE_MAX         : INT := 0;
    CONSTRAINED       : BOOLEAN := TRUE;
    INDEX_TYPE_DES    : ACCESS_TYPE_DESCRIPTOR := null;
    ARRAY_TYPE_DES    : ACCESS_TYPE_DESCRIPTOR := null;
    COUNT             : INTEGER := 0;

  begin
-- validate it and store necessary info to build it later
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_CONSTANT (VALID, "(", TRUE );
    GET_ARRAY_INDEX_TYPE (VALID, GOT_INDEX_TYPE, INDEX_TYPE, INDEX_TYPE_LAST,
                          RANGE_MIN, RANGE_MAX, INDEX_TYPE_DES);
    if DEBUGGING then
      PRINT_TO_FILE ("    array index type: " & BOOLEAN'IMAGE(VALID) &
                     " got index type: " & BOOLEAN'IMAGE(GOT_INDEX_TYPE));
      PRINT_TO_FILE ("    index type: " & INDEX_TYPE (1..INDEX_TYPE_LAST) &
                     " range min: " & INT'IMAGE(RANGE_MIN) & " max: " &
                     INT'IMAGE(RANGE_MAX));
      if INDEX_TYPE_DES /= null and then INDEX_TYPE_DES.FULL_NAME /= null then
        PRINT_TO_FILE ("    points to: " &
                       STRING(INDEX_TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all)
                       & "." & STRING(INDEX_TYPE_DES.FULL_NAME.NAME.all));
      end if;
    end if;
    GET_CONSTANT_MAYBE (VALID, NEED_RANGE, "RANGE", TRUE);
    GET_ARRAY_INDEX_RANGE (VALID, NEED_RANGE, GOT_RANGE, RANGE_LO,
                           RANGE_HI, GOT_INDEX_TYPE, RANGE_MIN, RANGE_MAX,
                           CONSTRAINED);
    if DEBUGGING then
      PRINT_TO_FILE ("    array index range - valid: " & BOOLEAN'IMAGE(VALID)
                     & " got range: " & BOOLEAN'IMAGE(GOT_RANGE) &
                     " range lo: " & INT'IMAGE(RANGE_LO) & " hi: " &
                     INT'IMAGE(RANGE_HI) & " constrained: " &
                     BOOLEAN'IMAGE(CONSTRAINED));
    end if;
    GET_CONSTANT (VALID, ")", TRUE);
    GET_CONSTANT (VALID, "OF", TRUE);
    GET_ARRAY_TYPE_OF (VALID, GOT_ARRAY_TYPE, ARRAY_TYPE, ARRAY_TYPE_LAST,
                       ARRAY_TYPE_DES);
    if DEBUGGING then
      PRINT_TO_FILE ("    array type of - valid: " & BOOLEAN'IMAGE(VALID) &
                     " got array type: " & BOOLEAN'IMAGE(GOT_ARRAY_TYPE) &
                     " array type: " & ARRAY_TYPE(1..ARRAY_TYPE_LAST));
      if ARRAY_TYPE_DES /= null and then ARRAY_TYPE_DES.FULL_NAME /= null then
        PRINT_TO_FILE ("    points to: " &
                     STRING(ARRAY_TYPE_DES.FULL_NAME.FULL_PACKAGE_NAME.all)
                     & "." & STRING(ARRAY_TYPE_DES.FULL_NAME.NAME.all));
```

```
      end if;
    end if;
    GET_CONSTANT (VALID, ";", FALSE);
    if GOT_INDEX_TYPE and not NEED_RANGE and not GOT_RANGE and CONSTRAINED then
      RANGE_LO := RANGE_MIN;
      RANGE_HI := RANGE_MAX;
    end if;
    if not VALID or else
           (NEED_RANGE and not GOT_RANGE) or else
           (not CONSTRAINED and not GOT_INDEX_TYPE) or else
           (not CONSTRAINED and RANGE_LO /= -1) or else
           (not CONSTRAINED and RANGE_HI /= -1) or else
           (not CONSTRAINED and not NEED_RANGE) then
      PRINT_ERROR ("Invalid type - array declaration");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      if DEBUGGING then
        PRINT_TO_FILE ("    valid: " & BOOLEAN'IMAGE(VALID) & " need range: "
                      & BOOLEAN'IMAGE(NEED_RANGE) & " got range: " &
                      BOOLEAN'IMAGE(GOT_RANGE) & " constrained: " &
                      BOOLEAN'IMAGE(CONSTRAINED) & " range lo: " &
                      INT'IMAGE(RANGE_LO) & " hi: " & INT'IMAGE(RANGE_HI));
      end if;
      return;
    end if;

-- build type descriptors here

    BUILD_STRING_TYPE_DESCRIPTORS (INDEX_TYPE_DES, ARRAY_TYPE_DES,
        CONSTRAINED, RANGE_LO, RANGE_HI, RANGE_MIN, RANGE_MAX, COUNT);
    if COUNT < 1 then
      PRINT_ERROR ("Invalid type - array declaration, no valid identifier");
    end if;
    if DEBUGGING then
      PRINT_TO_FILE ("    number of string type descriptors: " &
                    INTEGER'IMAGE(COUNT));
    end if;
  end PROCESS_ARRAY;

-------------------------------------------------------------------
--
-- GET_ARRAY_INDEX_TYPE
--
-- valid - if false on entry then don't do anything, don't alter
--         return false if we identify an attempt to define an array index
--            type and it's invalid.
--         do not alter if it's valid
--         We treat it as if we've found an identifier if it's alpha.
--         it must be a base type of integer and visible from our current
--            schema
```

```
--         if no identifier is found we use standard.integer as a default
-- got index type - true if we get one even if its the default
-- index type - identifier of the index type
-- index type last  - it's length.
-- range min - lo range from index type -1 if any integer is valid
-- range max - hi range from index type -1 if any integer is valid
-- index type des - pointer to type descriptor of index type, null if not here

  procedure GET_ARRAY_INDEX_TYPE
          (VALID              : in out BOOLEAN;
           GOT_INDEX_TYPE     : in out BOOLEAN;
           INDEX_TYPE         : in out STRING;
           INDEX_TYPE_LAST    : in out NATURAL;
           RANGE_MIN          : in out INT;
           RANGE_MAX          : in out INT;
           INDEX_TYPE_DES     : in out ACCESS_TYPE_DESCRIPTOR) is

    IDENT      : STRING (1..250) := (others => ' ');
    IDENT_LAST : NATURAL := 0;
    IDENT_DES  : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES   : ACCESS_FULL_NAME_DESCRIPTOR := null;
    IS_INT     : BOOLEAN := FALSE;
    LO_RANGE   : INT := 0;
    HI_RANGE   : INT := 0;
    ERROR      : INTEGER := 0;
    IDENT_DEF  : constant STRING := "STANDARD.INTEGER";
    DEFAULT    : BOOLEAN := FALSE;

  begin
    if VALID then
      INDEX_TYPE_DES := null;
      LOCATE_PREVIOUS_IDENTIFIER (TEMP_STRING (1..TEMP_STRING_LAST),
             TEMP_STRING_LAST, IDENT_DES, FULL_DES, ERROR, FALSE);
      if ERROR = 1 then
        DEFAULT := TRUE;
        IDENT_LAST := IDENT_DEF'LAST;
        IDENT (1..IDENT_LAST) := IDENT_DEF;
        LOCATE_PREVIOUS_IDENTIFIER (IDENT, IDENT_LAST, IDENT_DES, FULL_DES,
               ERROR, FALSE);
      end if;
      if ERROR = 1 then
        null;
      else
        GOT_INDEX_TYPE := TRUE;
        if DEFAULT then
          INDEX_TYPE_LAST := IDENT_DEF'LAST;
          INDEX_TYPE (1..INDEX_TYPE_LAST) := IDENT_DEF;
        else
          INDEX_TYPE_LAST := TEMP_STRING_LAST;
```

```
          INDEX_TYPE (1..INDEX_TYPE_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
          GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        end if;
        if ERROR > 1 then
          VALID := FALSE;
          PRINT_ERROR ("Invalid array index type - reference to unfound " &
                       "predefined identifier");
        else
          INDEX_TYPE_DES := FULL_DES.TYPE_IS;
          BASE_TYPE_INTEGER (FULL_DES, IS_INT, LO_RANGE, HI_RANGE);
          if not IS_INT then
            VALID := FALSE;
            PRINT_ERROR ("Invalid array index type - must " &
                         "have integer base");
          end if;
          RANGE_MIN := LO_RANGE;
          RANGE_MAX := HI_RANGE;
        end if;
      end if;
    end if;
  end GET_ARRAY_INDEX_TYPE;

-----------------------------------------------------------------
--
-- GET_ARRAY_INDEX_RANGE
--
-- if valid is false on entry then don't do anything
-- if need range then we have to find one or valid becomes false
-- set got range if we do find one
-- lo and hi range become the range specified, if got index type
-- is true then array lo and hi range better fall within the ranges on input,
-- if not valid = false.  If the range is <> then it's unconstrained and
-- we set the flag unconstrained as well as lo and hi to -1

  procedure GET_ARRAY_INDEX_RANGE
           (VALID              : in out BOOLEAN;
            NEED_RANGE         : in BOOLEAN;
            GOT_RANGE          : in out BOOLEAN;
            RANGE_LO           : in out INT;
            RANGE_HI           : in out INT;
            GOT_INDEX_TYPE     : in BOOLEAN;
            RANGE_MIN          : in out INT;
            RANGE_MAX          : in out INT;
            CONSTRAINED        : in out BOOLEAN) is

    RANGE1 : INT := 0;
    RANGE2 : INT := 0;
    OK     : BOOLEAN := FALSE;
```

```
begin
  if not VALID then
    return;
  end if;
  GOT_RANGE := FALSE;
  if TEMP_STRING (1..TEMP_STRING_LAST) = "<" then
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    if TEMP_STRING (1..TEMP_STRING_LAST) = ">" then
      CONSTRAINED := FALSE;
      GOT_RANGE := TRUE;
      RANGE_LO := -1;
      RANGE_HI := -1;
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      return;
    else
      VALID := FALSE;
      PRINT_ERROR ("Invalid array range definition - for " &
                   "unconstrained array");
      return;
    end if;
  end if;
  STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE1);
  if OK then
    GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
    GET_CONSTANT (VALID, ".", TRUE);
    if VALID then
      GET_CONSTANT (VALID, ".", TRUE);
      if VALID then
        STRING_TO_INT (TEMP_STRING (1..TEMP_STRING_LAST), OK, RANGE2);
        if OK then
          CONSTRAINED := TRUE;
          GOT_RANGE := TRUE;
          RANGE_LO := RANGE1;
          RANGE_HI := RANGE2;
          GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
          if GOT_INDEX_TYPE then
            if RANGE_MIN /= -1 or RANGE_MAX /= -1 then
              if RANGE_LO >= RANGE_MIN and RANGE_LO <= RANGE_MAX and
                 RANGE_HI >= RANGE_MIN and RANGE_HI <= RANGE_MAX then
                return;
              else
                VALID := FALSE;
                PRINT_ERROR ("Invalid array range definition - does not " &
                             "fall within parent's limits");
                return;
              end if;
            end if;
          end if;
        end if;
```

```
      end if;
    end if;
  else
    if not NEED_RANGE then
      OK := TRUE;
    end if;
  end if;
  if not OK or not VALID then
    VALID := FALSE;
    PRINT_ERROR ("Invalid array range definition - cannot determine range");
    return;
  end if;
  if NEED_RANGE then
    VALID := FALSE;
    PRINT_ERROR ("Invalid array range definition - range must be defined");
  end if;
end GET_ARRAY_INDEX_RANGE;

------------------------------------------------------------------------
--
-- GET_ARRAY_TYPE_OF
--
-- if valid is false return
-- got_array_type = true if we indeed have one
-- array_type will be the qualified identifier name of length array_type_last
-- array_type_des if the type descriptor
-- to be valid the array type identifier must be visible

  procedure GET_ARRAY_TYPE_OF
           (VALID            : in out BOOLEAN;
            GOT_ARRAY_TYPE   : in out BOOLEAN;
            ARRAY_TYPE       : in out STRING;
            ARRAY_TYPE_LAST  : in out NATURAL;
            ARRAY_TYPE_DES   : in out ACCESS_TYPE_DESCRIPTOR) is

    OK          : BOOLEAN := FALSE;
    IDENT       : STRING (1..250) := (others => ' ');
    PACK1       : STRING (1..250) := (others => ' ');
    PACK2       : STRING (1..250) := (others => ' ');
    IDENT_LAST  : NATURAL := 0;
    PACK1_LAST  : NATURAL := 0;
    PACK2_LAST  : NATURAL := 0;
    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    ERROR       : NATURAL := 0;

  begin
    if VALID then
      GOT_ARRAY_TYPE := FALSE;
```

```
      ARRAY_TYPE_DES := null;
      LOCATE_PREVIOUS_IDENTIFIER (TEMP_STRING (1..TEMP_STRING_LAST),
             TEMP_STRING_LAST, IDENT_DES, FULL_DES, ERROR, FALSE);
      if ERROR = 1 then
        PRINT_ERROR ("Invalid array type - must be defined");
        VALID := FALSE;
      else
        GOT_ARRAY_TYPE := TRUE;
        ARRAY_TYPE_LAST := TEMP_STRING_LAST;
        ARRAY_TYPE (1..ARRAY_TYPE_LAST) := TEMP_STRING (1..TEMP_STRING_LAST);
        GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
        if ERROR > 1 then
          VALID := FALSE;
          PRINT_ERROR ("Invalid array type - reference to unfound " &
                       "predefined identifier");
        else
          ARRAY_TYPE_DES := FULL_DES.TYPE_IS;
          if not BASE_TYPE_CHAR (FULL_DES) then
            VALID := FALSE;
            PRINT_ERROR ("Invalid array type - base type must be character");
          end if;
        end if;
      end if;
    end if;
  end GET_ARRAY_TYPE_OF;

------------------------------------------------------------------------
--
-- BUILD_STRING_TYPE_DESCRIPTORS
--

  procedure BUILD_STRING_TYPE_DESCRIPTORS
             (INDEX_TYPE_DES   : in ACCESS_TYPE_DESCRIPTOR;
              ARRAY_TYPE_DES   : in ACCESS_TYPE_DESCRIPTOR;
              CONSTRAINED      : in BOOLEAN;
              RANGE_LO         : in INT;
              RANGE_HI         : in INT;
              RANGE_MIN        : in INT;
              RANGE_MAX        : in INT;
              COUNT            : in out INTEGER) is

    NAME        : ACCESS_NAME_TO_PROCESS_LIST := FIRST_NAME_TO_PROCESS;
    IDENT_DES   : ACCESS_IDENTIFIER_DESCRIPTOR := null;
    FULL_DES    : ACCESS_FULL_NAME_DESCRIPTOR := null;
    STRING_DES  : ACCESS_STRING_DESCRIPTOR := null;
    LEN         : NATURAL := 0;

  begin
    COUNT := 0;
```

```
      while NAME /= null loop
        if VALID_NEW_IDENT_NAME (STRING (NAME.NAME.all)) then
          IDENT_DES := FIND_IDENTIFIER_DESCRIPTOR (STRING (NAME.NAME.all));
          ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS
             (IDENT_DES, FULL_DES, STRING(NAME.NAME.all));
          STRING_DES := GET_NEW_STRING_DESCRIPTOR;
          FULL_DES.TYPE_IS := STRING_DES;
          STRING_DES.WHICH_TYPE := STR_ING;
          STRING_DES.FULL_NAME := FULL_DES;
          STRING_DES.ULT_PARENT_TYPE := STRING_DES;
          STRING_DES.BASE_TYPE := STRING_DES;
          if CONSTRAINED then
            LEN := NATURAL (RANGE_HI - RANGE_LO + 1);
          else
            LEN := 0;
          end if;
          STRING_DES.LENGTH := LEN;
          STRING_DES.INDEX_TYPE := INDEX_TYPE_DES;
          STRING_DES.ARRAY_TYPE := ARRAY_TYPE_DES;
          STRING_DES.CONSTRAINED := CONSTRAINED;
          STRING_DES.ARRAY_RANGE_LO := RANGE_LO;
          STRING_DES.ARRAY_RANGE_HI := RANGE_HI;
          STRING_DES.ARRAY_RANGE_MIN := RANGE_MIN;
          STRING_DES.ARRAY_RANGE_MAX := RANGE_MAX;
          ADD_TYPE_DESCRIPTOR (STRING_DES);
          COUNT := COUNT + 1;
        else
          PRINT_ERROR ("Invalid identifier: " & STRING (NAME.NAME.all));
        end if;
        NAME := NAME.NEXT_NAME;
      end loop;
    end BUILD_STRING_TYPE_DESCRIPTORS;

end ARRAY_ROUTINES;
```

3.11.136 package ddl_type_spec.ada

```
with EXTRA_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES, LIST_ROUTINES,
     SUBROUTINES_2_ROUTINES, ARRAY_ROUTINES, INTEGER_ROUTINES, FLOAT_ROUTINES,
     ENUMERATION_ROUTINES, RECORD_ROUTINES, DERIVED_ROUTINES;
use  EXTRA_DEFINITIONS, SCHEMA_IO, SUBROUTINES_1_ROUTINES, LIST_ROUTINES,
     SUBROUTINES_2_ROUTINES, ARRAY_ROUTINES, INTEGER_ROUTINES, FLOAT_ROUTINES,
     ENUMERATION_ROUTINES, RECORD_ROUTINES, DERIVED_ROUTINES;

package TYPE_ROUTINES is

  procedure PROCESS_A_TYPE;

end TYPE_ROUTINES;
```

3.11.137 package ddl_type.ada

```
package body TYPE_ROUTINES is

-------------------------------------------------------------------
--
-- PROCESS_A_TYPE
--
-- first thing to do is store away the identifier or identifiers
-- then find out what type we're processing, array, integer, real or derived
-- then process accrodingly

  procedure PROCESS_A_TYPE is
  begin

-- first make chain of all identifiers returns with "is" in temp_string
    if DEBUGGING then
      PRINT_TO_FILE ("*** TYPE");
    end if;
    CURRENT_SCHEMA_UNIT.HAS_DECLARED_TYPES := TRUE;
    if CURRENT_SCHEMA_UNIT.IS_AUTH_PACKAGE then
      PRINT_ERROR ("Type declarations are not permitted within " &
                   "an authorization package");
    end if;
    if CURRENT_SCHEMA_UNIT.HAS_DECLARED_VARIABLES then
      PRINT_ERROR ("Type declarations must not be declared in a " &
                   "compilation unit which also");
      PRINT_TO_FILE ("   declares Ada/SQL program variables");
    end if;
    if not IN_ADA_SQL_PACKAGE then
      PRINT_ERROR ("Type declarations permitted only within the ADA_SQL " &
                   " subpackages");
      FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      return;
    end if;
    if MAKE_LIST_OF_NAMES then
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if TEMP_STRING(1..TEMP_STRING_LAST) = "ARRAY" then
        if DEBUGGING then
          PRINT_TO_FILE ("    ARRAY");
        end if;
        PROCESS_ARRAY;
      elsif TEMP_STRING(1..TEMP_STRING_LAST) = "RANGE" then
        if DEBUGGING then
          PRINT_TO_FILE ("    INTEGER");
        end if;
        PROCESS_INTEGER;
      elsif TEMP_STRING(1..TEMP_STRING_LAST) = "DIGITS" then
        if DEBUGGING then
```

```
          PRINT_TO_FILE ("    FLOAT");
        end if;
        PROCESS_FLOAT;
      elsif TEMP_STRING(1..TEMP_STRING_LAST) = "(" then
        if DEBUGGING then
          PRINT_TO_FILE ("    ENUMERATION");
        end if;
        PROCESS_ENUMERATION;
      elsif TEMP_STRING(1..TEMP_STRING_LAST) = "RECORD" then
        if DEBUGGING then
          PRINT_TO_FILE ("    RECORD");
        end if;
        PROCESS_RECORD;
      elsif TEMP_STRING(1..TEMP_STRING_LAST) = "NEW" then
        if DEBUGGING then
          PRINT_TO_FILE ("    DERIVED");
        end if;
        PROCESS_DERIVED;
      else
        PRINT_ERROR ("Invalid type declaration");
        FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
      end if;
    end if;
    FIND_END_OF_STATEMENT (TEMP_STRING, TEMP_STRING_LAST);
  end PROCESS_A_TYPE;

end TYPE_ROUTINES;
```

3.11.138 package ddl_driver_spec.ada

```
with DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES,
     SUBROUTINES_4_ROUTINES, WITH_ROUTINES, USE_ROUTINES,
     PACKAGE_ROUTINES, END_ROUTINES, TYPE_ROUTINES, SUBTYPE_ROUTINES,
     FUNCTION_ROUTINES, SCHEMA_AUTHORIZATION_ROUTINES, VARIABLE_ROUTINES;
use  DDL_DEFINITIONS, EXTRA_DEFINITIONS, IO_DEFINITIONS, SCHEMA_IO,
     GET_NEW_DESCRIPTOR_ROUTINES, ADD_DESCRIPTOR_ROUTINES,
     SEARCH_DESCRIPTOR_ROUTINES, SUBROUTINES_1_ROUTINES,
     SUBROUTINES_4_ROUTINES, WITH_ROUTINES, USE_ROUTINES,
     PACKAGE_ROUTINES, END_ROUTINES, TYPE_ROUTINES, SUBTYPE_ROUTINES,
     FUNCTION_ROUTINES, SCHEMA_AUTHORIZATION_ROUTINES, VARIABLE_ROUTINES;

package DRIVER is

  procedure PROCESS_SCHEMA_UNIT
          (USER_SCHEMA_UNIT : in out STRING);

  procedure PROCESS_FULL_SCHEMA_UNIT
          (NAME : in STRING);
```

```
  procedure SET_UP_CURRENT_SCHEMA_UNIT
           (NAME : in STRING);

  procedure WHICH_PROCESS
           (THE_PROCESS : in STRING;
            SCHEMA      : in ACCESS_SCHEMA_UNIT_DESCRIPTOR);

end DRIVER;
```

3.11.139 package ddl_driver.ada

```
package body DRIVER is

-----------------------------------------------------------------
--
-- PROCESS_SCHEMA_UNIT

  procedure PROCESS_SCHEMA_UNIT
           (USER_SCHEMA_UNIT : in out STRING) is

    LENGTH     : NATURAL := 0;

  begin
    LENGTH := USER_SCHEMA_UNIT'LAST;
    ADJUST_USER_SCHEMA (USER_SCHEMA_UNIT, LENGTH);
    if LENGTH < 1 then
      PRINT_MESSAGE ("Invalid schema unit name, try again.");
      return;
    end if;
    OPEN_OUTPUT_FILE (USER_SCHEMA_UNIT);
    if not CALLED_STANDARD_YET then
      PROCESS_FULL_SCHEMA_UNIT  (STANDARD_NAME);
      PROCESS_FULL_SCHEMA_UNIT  (DATABASE_NAME);
      PROCESS_FULL_SCHEMA_UNIT  (CURSOR_NAME);
      CALLED_STANDARD_YET := TRUE;
    end if;
    PROCESS_FULL_SCHEMA_UNIT  (USER_SCHEMA_UNIT);
  end PROCESS_SCHEMA_UNIT;

-----------------------------------------------------------------
--
-- PROCESS_FULL_SCHEMA_UNIT
--
-- set up the current schema unit, which might be a new one or one
-- already done or one currently in process.
-- we loop doing the following until reaching the end of a file
--    then till exhausting the yet to do list
--
-- read the next token, which must be something we recgonise.
-- when the end of the file is reached the DONE flag is set
```

```
-- if we are already in the middle of withing, flag set, then we call
--    PROCESS_WITH to do the next with in line or look for ; as a clue to the
--    end of withing
-- if the token is use, package, end, type, subtype, function, or
--    schema_authorization we have special routines to process the whole
--    statement
-- if the token is anything else tell the user it's an error

  procedure PROCESS_FULL_SCHEMA_UNIT
           (NAME : in STRING) is
  begin
    if DEBUGGING then
      PRINT_TO_FILE ("*** User request processing schema unit: "
                     & NAME);
    end if;
    SET_UP_CURRENT_SCHEMA_UNIT (NAME);
    while CURRENT_SCHEMA_UNIT /= null loop
      while CURRENT_SCHEMA_UNIT /= null and then
            CURRENT_SCHEMA_UNIT.SCHEMA_STATUS /= PROCESSING and then
            CURRENT_SCHEMA_UNIT.SCHEMA_STATUS /= WITHING loop
        CURRENT_SCHEMA_UNIT := FIND_NEXT_YET_TO_DO_DESCRIPTOR;
      end loop;
      exit when CURRENT_SCHEMA_UNIT = null;
      if DEBUGGING then
        PRINT_TO_FILE ("*** Processing from schema unit: " &
                       STRING(CURRENT_SCHEMA_UNIT.NAME.all));
        PRINT_TO_FILE ("                 current package: " &
                       OUR_PACKAGE_NAME (1..OUR_PACKAGE_NAME_LAST));
      end if;
      GET_STRING (CURRENT_SCHEMA_UNIT, TEMP_STRING, TEMP_STRING_LAST);
      if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS < DONE then
        WHICH_PROCESS (TEMP_STRING (1..TEMP_STRING_LAST), CURRENT_SCHEMA_UNIT);
        case CURRENT_PROCESS is
          when ITS_ALREADY_WITHING      => PROCESS_WITH;
          when ITS_WITH                 => PROCESS_WITH;
          when ITS_USE                  => PROCESS_USE;
          when ITS_PACKAGE              => PROCESS_PACKAGE;
          when ITS_END                  => PROCESS_END;
          when ITS_TYPE                 => PROCESS_A_TYPE;
          when ITS_SUBTYPE              => PROCESS_SUBTYPE;
          when ITS_FUNCTION             => PROCESS_FUNCTION;
          when ITS_SCHEMA_AUTHORIZATION => PROCESS_SCHEMA_AUTHORIZATION;
          when ITS_EOL                  => null;
          when ITS_UNKNOWN              => TRY_TO_PROCESS_VARIABLE;
          when ITS_FINISHED             => null;
        end case;
      end if;
    end loop;
  end PROCESS_FULL_SCHEMA_UNIT;
```

```
-----------------------------------------------------------------------
--
-- SET_UP_CURRENT_SCHEMA_UNIT
--
-- set up the current schema, either an old one that wasn't finished or a
-- new one in which case we have to open the file.
-- search the list of already done schema_units, if this one hasn't
-- been done set up new pointers for it, add it to the chain and
-- set the name and open an input stream.
-- and if it's not DDL_STANDARD_FOR_ADA_SQL then show withing and using of it

  procedure SET_UP_CURRENT_SCHEMA_UNIT
           (NAME : in STRING) is
    NAME_UP : STRING (1..100) := (others => ' ');
    NAME_UP_LEN : INTEGER := 1;
  begin
    NAME_UP_LEN := NAME'LENGTH;
    NAME_UP (1..NAME_UP_LEN) := NAME;
    UPPER_CASE (NAME_UP (1..NAME_UP_LEN));
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR
                     (NAME_UP (1..NAME_UP_LEN));
    if CURRENT_SCHEMA_UNIT = null then
      CURRENT_SCHEMA_UNIT := GET_NEW_SCHEMA_UNIT_DESCRIPTOR;
      ADD_SCHEMA_UNIT_DESCRIPTOR (CURRENT_SCHEMA_UNIT);
      CURRENT_SCHEMA_UNIT.NAME := GET_NEW_LIBRARY_UNIT_NAME (NAME);
      OPEN_SCHEMA_UNIT (CURRENT_SCHEMA_UNIT);
      UPPER_CASE (STRING (CURRENT_SCHEMA_UNIT.NAME.all));
      SET_UP_WITH_USE_STANDARD_FOR_SCHEMA (CURRENT_SCHEMA_UNIT);
    end if;
    SET_UP_OUR_PACKAGE_NAME;
  end SET_UP_CURRENT_SCHEMA_UNIT;

-----------------------------------------------------------------
--
-- WHICH_PROCESS
--
-- we're given a token and the schema we're processing, we want to return an
-- enumeration type for which process to do

  procedure WHICH_PROCESS
           (THE_PROCESS : in STRING;
            SCHEMA      : in ACCESS_SCHEMA_UNIT_DESCRIPTOR) is
  begin
    if CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = WITHING then
      CURRENT_PROCESS := ITS_ALREADY_WITHING;
    elsif CURRENT_SCHEMA_UNIT.SCHEMA_STATUS = DONE then
      CURRENT_PROCESS := ITS_FINISHED;
    elsif THE_PROCESS = "WITH" then
      CURRENT_PROCESS := ITS_WITH;
```

```
    elsif THE_PROCESS = "USE" then
      CURRENT_PROCESS := ITS_USE;
    elsif THE_PROCESS = "PACKAGE" then
      CURRENT_PROCESS := ITS_PACKAGE;
    elsif THE_PROCESS = "END" then
      CURRENT_PROCESS := ITS_END;
    elsif THE_PROCESS = "TYPE" then
      CURRENT_PROCESS := ITS_TYPE;
    elsif THE_PROCESS = "SUBTYPE" then
      CURRENT_PROCESS := ITS_SUBTYPE;
    elsif THE_PROCESS = "FUNCTION" then
      CURRENT_PROCESS := ITS_FUNCTION;
    elsif THE_PROCESS = "SCHEMA_AUTHORIZATION" then
      CURRENT_PROCESS := ITS_SCHEMA_AUTHORIZATION;
    elsif THE_PROCESS = ";" then
      CURRENT_PROCESS := ITS_EOL;
    else
      CURRENT_PROCESS := ITS_UNKNOWN;
    end if;
  end WHICH_PROCESS;

end DRIVER;
```

3.11.140 package ddl_call_to_ddl_spec.ada

```
with IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SEARCH_DESCRIPTOR_ROUTINES, DRIVER, SUBROUTINES_1_ROUTINES;
use  IO_DEFINITIONS, DDL_DEFINITIONS, EXTRA_DEFINITIONS, SCHEMA_IO,
     SEARCH_DESCRIPTOR_ROUTINES, DRIVER, SUBROUTINES_1_ROUTINES;

package CALL_TO_DDL_ROUTINES is

  procedure CALL_TO_DDL_OPEN_SCHEMA_UNIT
            (USER_SCHEMA_UNIT : in STRING);

  procedure CALL_TO_DDL_WITH
            (NAME : in STRING);

  procedure CALL_TO_DDL_USE
            (NAME : in STRING);

  procedure CALL_TO_DDL_CLOSE;

end CALL_TO_DDL_ROUTINES;
```

3.11.141 package ddl_call_to_ddl.ada

```
package body CALL_TO_DDL_ROUTINES is

-----------------------------------------------------------------
```

```
--
-- CALL_TO_DDL_OPEN_SCHEMA_UNIT

  procedure CALL_TO_DDL_OPEN_SCHEMA_UNIT
            (USER_SCHEMA_UNIT : in STRING) is

    NAME        : STRING (1..USER_SCHEMA_UNIT'LAST) := USER_SCHEMA_UNIT;
    LEN         : NATURAL := 0;

  begin
    WHERE_IS_SCHEMA_FROM := CALLS;
    LEN := NAME'LAST;
    ADJUST_USER_SCHEMA (NAME, LEN);
    SCHEMA_UNIT_CALLED_LEN := LEN;
    SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN) := NAME (1..LEN);
    PROCESS_SCHEMA_UNIT (NAME);
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (NAME (1..LEN));
  end CALL_TO_DDL_OPEN_SCHEMA_UNIT;

-----------------------------------------------------------------------
--
-- CALL_TO_DDL_WITH

  procedure CALL_TO_DDL_WITH
            (NAME : in STRING) is

    SCHEMA_NAME        : STRING (1..SCHEMA_UNIT_CALLED_LEN) :=
                         SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN);
  begin

    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
    if CURRENT_SCHEMA_UNIT = null or WHERE_IS_SCHEMA_FROM /= CALLS then
      PRINT_ERROR ("Calling call_to_ddl_with without first " &
                   "calling call_to_ddl_open_schema");
      return;
    end if;
    CURRENT_SCHEMA_UNIT.SCHEMA_STATUS := PROCESSING;
    CURRENT_SCHEMA_UNIT.STREAM.LAST := NAME'LAST + 6;
    CURRENT_SCHEMA_UNIT.STREAM.ORIG_BUF (1..CURRENT_SCHEMA_UNIT.STREAM.LAST)
                         := ("WITH " & NAME & ";");
    CURRENT_SCHEMA_UNIT.STREAM.NEXT := 1;
    CURRENT_SCHEMA_UNIT.STREAM.LINE := CURRENT_SCHEMA_UNIT.STREAM.LINE + 1;
    CURRENT_SCHEMA_UNIT.STREAM.BUFFER (1..CURRENT_SCHEMA_UNIT.STREAM.LAST)
                         := ("WITH " & NAME & ";");
    UPPER_CASE (CURRENT_SCHEMA_UNIT.STREAM.ORIG_BUF
                (1..CURRENT_SCHEMA_UNIT.STREAM.LAST));
    PROCESS_SCHEMA_UNIT (SCHEMA_NAME);
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
  end CALL_TO_DDL_WITH;
```

```
-----------------------------------------------------------------------
--
-- CALL_TO_DDL_USE

  procedure CALL_TO_DDL_USE
            (NAME : in STRING) is

    SCHEMA_NAME        : STRING (1..SCHEMA_UNIT_CALLED_LEN) :=
                         SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN);
  begin
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
    if CURRENT_SCHEMA_UNIT = null or WHERE_IS_SCHEMA_FROM /= CALLS then
      PRINT_ERROR ("Calling call_to_ddl_use without first " &
                   "calling call_to_ddl_open_schema");
      return;
    end if;
    CURRENT_SCHEMA_UNIT.SCHEMA_STATUS := PROCESSING;
    CURRENT_SCHEMA_UNIT.STREAM.LAST := NAME'LAST + 5;
    CURRENT_SCHEMA_UNIT.STREAM.ORIG_BUF (1..CURRENT_SCHEMA_UNIT.STREAM.LAST)
                       := ("USE " & NAME & ";");
    CURRENT_SCHEMA_UNIT.STREAM.NEXT := 1;
    CURRENT_SCHEMA_UNIT.STREAM.LINE := CURRENT_SCHEMA_UNIT.STREAM.LINE + 1;
    CURRENT_SCHEMA_UNIT.STREAM.BUFFER (1..CURRENT_SCHEMA_UNIT.STREAM.LAST)
                       := ("USE " & NAME & ";");
    UPPER_CASE (CURRENT_SCHEMA_UNIT.STREAM.BUFFER
                (1..CURRENT_SCHEMA_UNIT.STREAM.LAST));
    PROCESS_SCHEMA_UNIT (SCHEMA_NAME);
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
  end CALL_TO_DDL_USE;

-----------------------------------------------------------------------
--
--  CALL_TO_DDL_CLOSE

  procedure CALL_TO_DDL_CLOSE is

    SCHEMA_NAME        : STRING (1..SCHEMA_UNIT_CALLED_LEN) :=
                         SCHEMA_UNIT_CALLED (1..SCHEMA_UNIT_CALLED_LEN);
  begin
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
    CLOSE_OUTPUT_FILE;
    CURRENT_SCHEMA_UNIT := FIND_SCHEMA_UNIT_DESCRIPTOR (SCHEMA_NAME);
  end CALL_TO_DDL_CLOSE;

end CALL_TO_DDL_ROUTINES;
```

3.11.142 package scanb.ada

```
-- scanb.ada - driver for DML processing of Ada/SQL Application Scanner
```

```
with LEXICAL_ANALYZER, CALL_TO_DDL_ROUTINES, SYNTACTICALLY, POST_PROCESS,
 STATEMENT, SELECT_STATEMENT;
with SHOW_ROUTINES, EXTRA_DEFINITIONS;  --% for debug
use  LEXICAL_ANALYZER;
package body SCAN_DML is

   type ADA_SQL_KEYWORD is
      (CLOSE, DECLAR, DELETE_FROM, FETCH, INSERT_INTO, OPEN, SELEC, SELECT_ALL,
       SELECT_DISTINCT, UPDATE);

-- For the time being, the following routines are stubbed until they are
-- implemented (probably in other packages).
--procedure PROCESS_OPEN            is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_DECLAR          is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_DELETE_FROM     is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_FETCH           is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_INSERT_INTO     is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_CLOSE           is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_SELEC           is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_SELEC_ALL       is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_SELEC_DISTINCT is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_UPDATE          is begin EAT_NEXT_TOKEN; end;
--procedure PROCESS_PACKAGE         is begin EAT_NEXT_TOKEN; end;

procedure PROCESS_WITH_CLAUSE is
   TOKEN : LEXICAL_TOKEN := FIRST_LOOK_AHEAD_TOKEN;
begin
   if TOKEN.KIND /= RESERVED_WORD or else TOKEN.RESERVED_WORD /= R_WITH then
      REPORT_SYSTEM_ERROR (TOKEN, "Expecting WITH");
   end if;
   EAT_NEXT_TOKEN;
   loop
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND /= IDENTIFIER then
         REPORT_SYNTAX_ERROR (TOKEN, "Expecting library unit name");
      end if;
      CALL_TO_DDL_ROUTINES.CALL_TO_DDL_WITH (TOKEN.ID.all);
      EAT_NEXT_TOKEN;
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
   exit when TOKEN.KIND = DELIMITER and then TOKEN.DELIMITER = SEMICOLON;
      SYNTACTICALLY.PROCESS_DELIMITER (COMMA);
   end loop;
   SYNTACTICALLY.PROCESS_DELIMITER (SEMICOLON);
-- exception
--   when others =>
--      REPORT_FATAL_ERROR
--         (FIRST_LOOK_AHEAD_TOKEN, "Error while processing WITH clause");
end PROCESS_WITH_CLAUSE;
```

```
procedure PROCESS_USE_CLAUSE is
   TOKEN : LEXICAL_TOKEN := FIRST_LOOK_AHEAD_TOKEN;
begin
   if TOKEN.KIND /= RESERVED_WORD or else TOKEN.RESERVED_WORD /= R_USE then
      REPORT_SYSTEM_ERROR (TOKEN, "Expecting USE");
   end if;
   EAT_NEXT_TOKEN;
   loop
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND /= IDENTIFIER then
         REPORT_SYNTAX_ERROR (TOKEN, "Expecting library unit name");
      end if;
      CALL_TO_DDL_ROUTINES.CALL_TO_DDL_USE (TOKEN.ID.all);
      EAT_NEXT_TOKEN;
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
   exit when TOKEN.KIND = DELIMITER and then TOKEN.DELIMITER = SEMICOLON;
      SYNTACTICALLY.PROCESS_DELIMITER (COMMA);
   end loop;
   SYNTACTICALLY.PROCESS_DELIMITER (SEMICOLON);
--exception
--   when others =>
--      REPORT_FATAL_ERROR
--         (FIRST_LOOK_AHEAD_TOKEN, "Error while processing USE clause");
end PROCESS_USE_CLAUSE;

procedure PROCESS_CONTEXT_CLAUSE is
   TOKEN : LEXICAL_TOKEN;
begin
   -- An application compilation unit which contains Ada/SQL DML statements
   -- must have at least one WITH clause which identifies the Ada/SQL DDL
   -- units.  This DDL WITH clause must be the first WITH clause in the
   -- compilation unit and may be optionally followed by a USE clause for
   -- the DDL units.  An application programmer is free to provide subsequent
   -- WITH and USE clauses which can name other non-Ada/SQL library units.
   TOKEN := FIRST_LOOK_AHEAD_TOKEN;
   if TOKEN.KIND = RESERVED_WORD and then
      TOKEN.RESERVED_WORD = R_WITH then
      PROCESS_WITH_CLAUSE;
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = RESERVED_WORD and then
         TOKEN.RESERVED_WORD = R_USE then
         PROCESS_USE_CLAUSE;
      end if;
   end if;
   -- Skip over any subsequent WITH or USE clauses.
   loop
      TOKEN := FIRST_LOOK_AHEAD_TOKEN;
      if TOKEN.KIND = END_OF_FILE then
         REPORT_FATAL_ERROR
```

```
            (TOKEN, "End of file without PACKAGE, PROCEDURE, or FUNCTION seen");
      end if;
   exit when TOKEN.KIND = RESERVED_WORD and then
      (TOKEN.RESERVED_WORD = R_PACKAGE or
       TOKEN.RESERVED_WORD = R_PROCEDURE or
       TOKEN.RESERVED_WORD = R_FUNCTION);
      EAT_NEXT_TOKEN;
   end loop;
end PROCESS_CONTEXT_CLAUSE;

procedure PROCESS_LIBRARY_UNIT_BODY is
   -- BNF:  library_unit_body ::= subprogram_body  |  package_body
   --       package_body      ::= package body package_simple_name rest_of_body
   --       subprogram_body   ::= procedure identifier rest_of_body |
   --                             function identifier rest_of_body
   --       rest_of_body      ::= {text} [dml_statement {text}]
   --       dml_statement     ::= close_statement    |
   --                             commit_statement   |
   --                             declare_cursor     |
   --                             delete_statement   |
   --                             fetch_statement    |
   --                             insert_statement   |
   --                             open_statement     |
   --                             rollback_statement |
   --                             select_statement   |
   --                             update_statement
   TOKEN : LEXICAL_TOKEN := NEXT_TOKEN;

   procedure GET_KEYWORD
      (WORD  : in  STRING;
       KEY   : out ADA_SQL_KEYWORD;
       FOUND : out BOOLEAN) is
   begin
      KEY := ADA_SQL_KEYWORD'VALUE(WORD);
      FOUND := TRUE;
   exception
      when others => FOUND := FALSE;
   end GET_KEYWORD;

begin
   if TOKEN.KIND /= RESERVED_WORD then
      REPORT_SYNTAX_ERROR (TOKEN, "Expecting PACKAGE, PROCEDURE, or FUNCTION");
   end if;
   case TOKEN.RESERVED_WORD is
      when R_PACKAGE | R_PROCEDURE | R_FUNCTION => null;
      when others =>
         REPORT_SYNTAX_ERROR (TOKEN, "Expecting PACKAGE, PROCEDURE, or FUNCTION");
   end case;
   if TOKEN.RESERVED_WORD = R_PACKAGE then
```

```
      TOKEN := NEXT_TOKEN;
      if TOKEN.KIND /= RESERVED_WORD or else
         TOKEN.RESERVED_WORD /= R_BODY then
         REPORT_SYNTAX_ERROR (TOKEN, "Expecting reserved word BODY");
      end if;
   end if;
   TOKEN := NEXT_TOKEN;
   if TOKEN.KIND /= IDENTIFIER then
      REPORT_SYNTAX_ERROR (TOKEN, "Expecting body name");
   end if;
   loop
      begin
         TOKEN := FIRST_LOOK_AHEAD_TOKEN;
         case TOKEN.KIND is
            when END_OF_FILE =>
               exit;
            when IDENTIFIER =>
               declare
                  KEYWORD : ADA_SQL_KEYWORD;
                  FOUND   : BOOLEAN;
               begin
                  GET_KEYWORD (TOKEN.ID.all, KEYWORD, FOUND);
                  if FOUND then
                     case KEYWORD is
                        when CLOSE =>
                         STATEMENT.PROCESS_CLOSE_STATEMENT;
                        when DECLAR =>
                         SELECT_STATEMENT.PROCESS_DECLARE_CURSOR;
                        when DELETE_FROM =>
                         STATEMENT.PROCESS_DELETE_STATEMENT_SEARCHED;
                        when FETCH =>
                         SELECT_STATEMENT.PROCESS_FETCH;
                        when INSERT_INTO =>
                         SELECT_STATEMENT.PROCESS_INSERT_INTO;
                        when OPEN =>
                         STATEMENT.PROCESS_OPEN_STATEMENT;
                        when SELEC | SELECT_ALL | SELECT_DISTINCT =>
                         SELECT_STATEMENT.PROCESS_SELECT_STATEMENT;
                        when UPDATE =>
                         STATEMENT.PROCESS_UPDATE_STATEMENT_SEARCHED;
                        when others          => EAT_NEXT_TOKEN;
                     end case;
                  else
                     EAT_NEXT_TOKEN;
                  end if;
               end;
            when RESERVED_WORD =>
               if TOKEN.RESERVED_WORD = R_PACKAGE then
                  STATEMENT.PROCESS_PACKAGE;
```

```
                else
                   EAT_NEXT_TOKEN;
                end if;
             when others =>
                EAT_NEXT_TOKEN;
          end case;
       exception
          when SYNTAX_ERROR =>
             --% for debug
             REPORT_NOTE (FIRST_LOOK_AHEAD_TOKEN, "Syntax error detected");
             --% end debug
             loop
                TOKEN := FIRST_LOOK_AHEAD_TOKEN;
                case TOKEN.KIND is
                   when DELIMITER   => exit when TOKEN.DELIMITER = SEMICOLON;
                   when END_OF_FILE => exit;
                   when others      => null;
                end case;
                EAT_NEXT_TOKEN;
             end loop;
       end;
    end loop;
 end PROCESS_LIBRARY_UNIT_BODY;

 procedure PROCESS_SECONDARY_UNIT is
    -- BNF:  secondary_unit ::= library_unit_body
 begin
    PROCESS_LIBRARY_UNIT_BODY;
 end PROCESS_SECONDARY_UNIT;

 procedure PROCESS_COMPILATION_UNIT is
    -- BNF:  compilation      ::= compilation_unit
    -- BNF:  compilation_unit ::= context_caluse secondary_unit
 begin
    PROCESS_CONTEXT_CLAUSE;
    PROCESS_SECONDARY_UNIT;
 end PROCESS_COMPILATION_UNIT;

 function GET_COMPILATION_UNIT_NAME
    return STRING is
    LOOK_AHEAD : LEXICAL_TOKEN;
 begin
    LOOK_AHEAD := FIRST_LOOK_AHEAD_TOKEN;
    while LOOK_AHEAD.KIND /= END_OF_FILE and then
       (LOOK_AHEAD.KIND /= RESERVED_WORD or else
        (LOOK_AHEAD.RESERVED_WORD /= R_PACKAGE and
         LOOK_AHEAD.RESERVED_WORD /= R_PROCEDURE and
         LOOK_AHEAD.RESERVED_WORD /= R_FUNCTION)) loop
       LOOK_AHEAD := NEXT_LOOK_AHEAD_TOKEN;
```

```
      end loop;
      if LOOK_AHEAD.KIND = END_OF_FILE then
         REPORT_FATAL_ERROR
            (LOOK_AHEAD,
             "End of file encountered without PACKAGE, PROCEDURE, or FUNCTION seen"
      end if;
      if LOOK_AHEAD.RESERVED_WORD = R_PACKAGE then
         -- reserved word BODY should be next.
         LOOK_AHEAD := NEXT_LOOK_AHEAD_TOKEN;
         if LOOK_AHEAD.KIND /= RESERVED_WORD or else
            LOOK_AHEAD.RESERVED_WORD /= R_BODY then
            REPORT_SYNTAX_ERROR
               (LOOK_AHEAD,
                "Expecting reserved word BODY");
         end if;
      end if;
      LOOK_AHEAD := NEXT_LOOK_AHEAD_TOKEN;
      if LOOK_AHEAD.KIND /= IDENTIFIER then
         REPORT_SYNTAX_ERROR
            (LOOK_AHEAD,
             "Expecting body name");
      end if;
      return LOOK_AHEAD.ID.all;
   exception
      when SYNTAX_ERROR =>
         REPORT_FATAL_ERROR ("Cannot determine compilation unit name");
   end GET_COMPILATION_UNIT_NAME;

   procedure PROCESS_APPLICATION_UNIT
      (UNIT_FILENAME             : in STRING;
       LISTING_FILENAME          : in STRING := "";
       GENERATED_PACKAGE_FILENAME : in STRING) is

   begin
   --% for debug
      --EXTRA_DEFINITIONS.DEBUGGING := FALSE;
   --% end debug
      -- Open the application compilation unit.
      LEXICAL_ANALYZER.OPEN_TOKEN_STREAM (UNIT_FILENAME, LISTING_FILENAME);
      -- Initialize the DDL reader.  Note that the DDL reader requires the name
      -- of the application compilation unit to initialize its data structures.
      -- Until the design can be changed to remove this requirement, we will use
      -- the token look-ahead facility of the Lexical Analyzer to retrieve the
      -- compilation unit name.
   --   CALL_TO_DDL_ROUTINES.INITIALIZE_FOR_DML_UNIT
      CALL_TO_DDL_ROUTINES.CALL_TO_DDL_OPEN_SCHEMA_UNIT
         (GET_COMPILATION_UNIT_NAME);
      PROCESS_COMPILATION_UNIT;
   --% for debug
```

```
--    SHOW_ROUTINES.SHOW_DATA;
--% end debug
   CALL_TO_DDL_ROUTINES.CALL_TO_DDL_CLOSE;
   LEXICAL_ANALYZER.CLOSE_TOKEN_STREAM;
--    if LEXICAL_ANALYZER.SEVERE_ERRORS = 0 then
      POST_PROCESS.GENERATE_PACKAGE (GENERATED_PACKAGE_FILENAME);
--    end if;
   LEXICAL_ANALYZER.PRODUCE_ERROR_LISTING;
exception
   when FATAL_ERROR =>
      LEXICAL_ANALYZER.CLOSE_TOKEN_STREAM;
      LEXICAL_ANALYZER.PRODUCE_ERROR_LISTING;
--    when others =>
--       begin
--          LEXICAL_ANALYZER.REPORT_FATAL_ERROR ("Unexpected exception occurred.");
--       exception
--          when others => null;
--       end;
--       LEXICAL_ANALYZER.CLOSE_TOKEN_STREAM;
--       LEXICAL_ANALYZER.PRODUCE_ERROR_LISTING;
end PROCESS_APPLICATION_UNIT;

end SCAN_DML;
```

Distribution List for IDA Memorandum Report M-460

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
Capt Steve Myatt WIS JPMO/XPT Washington, D.C. 20330-6600	2 copies
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22314	1 copy
Ms. Patty Hicks DSAC-SR 3990 Broad St. Columbus, OH 43216-5002	1 copy
Ms. Elinor Koffee DSAC-SR 3990 Broad St. Columbus, OH 43216-5002	1 copy
Ms. Kerry Hilliard 7321 Franklin Road Annandale, VA 22003	1 copy
CSED Review Panel	
Dr. Dan Alpert, Director Center for Advanced Study University of Illinois 912 W. Illinois Street Urbana, Illinois 61801	1 copy
Dr. Barry W. Boehm TRW Defense Systems Group MS 2-2304 One Space Park Redondo Beach, CA 90278	1 copy
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1 copy

NAME AND ADDRESS	NUMBER OF COPIES
Dr. Larry E. Druffel Software Engineering Institute Shadyside Place 480 South Aiken Av. Pittsburgh, PA 15231	1 copy
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1 copy
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1 copy
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1 copy
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1 copy
IDA	
General W.Y. Smith, HQ	1 copy
Mr. Seymour Deitchman, HQ	1 copy
Mr. Philip Major, HQ	1 copy
Ms. Karen H. Weber, HQ	1 copy
Dr. Jack Kramer, CSED	1 copy
Dr. Robert I. Winner, CSED	1 copy
Dr. John Salasin, CSED	1 copy
Mr. Bill Brykczynski, CSED	5 copies
Ms. Audrey A. Hook, CSED	1 copy
Ms. Katydean Price, CSED	2 copies
IDA Control & Distribution Vault	3 copies

END
DATED
FILM
8-88
DTIC